Properties and Algorithms of the (n, k)-Arrangement Graphs and Augmented Cubes

Nafiseh Motevallibashi
Department of Computer Science

Submitted in partial fulfilment
of the requirements for the degree of

Master of Science

Faculty of Mathematics and Science
Brock University
St. Catharines, Ontario

Abstract

The (n, k)-arrangement graph was first introduced in 1992 as a generalization of the star graph topology. Choosing an arrangement topology is more efficient in comparison with a star graph as we can have a closer number of nodes to what is needed. Also it has other advantages such as a lower degree and a smaller diameter, depending on k. In this thesis we investigate the problem of finding $k(n-k)$ disjoint paths from a source node to $k(n-k)$ target nodes in an (n, k)-arrangement interconnection network such that no path has length more than $diameter + (n-k) + 2$, where *diameter* is the maximum length of shortest path between any two nodes in the graph. These disjoint paths are built by routing to all neighbors of the source node and fixing specific elements in each of the k positions of the node representation in an (n, k)-arrangement graph. Moreover, a simple routing is presented for finding n disjoint paths between two nodes which are located in different sub-graphs. The lengths are no more than $d(t, s) + 4$, for $d(t, s)$ being the shortest path length between two nodes s and t. This routing algorithm needs $O(n^2)$ time to find all n these paths.

In addition to arrangement graphs, we also study augmented cubes, first introduced in 2002, a desirable variation of the hypercube. An augmented cube of dimension n has a higher degree and a lower diameter in comparison with the hypercube. We introduce an $O(n^3)$ algorithm for finding disjoint shortest paths from a single source node to $2n-1$ different target nodes.

Acknowledgement

Foremost, I would like to express my sincere gratitude to my supervisor Dr. Ke Qiu for his unfailing support, patience and encouragement through my studies. The door of Dr. Qiu office was always open whenever I had a question or I ran into a trouble. He led me in the right direction and allowed this thesis be my own work. I could not have imagined having a better supervisor and mentor for my master study.

Besides my supervisor, I would also like to thank the rest of my thesis committee: Dr. Michael Winter, Dr. Sheridan Houghton, and Dr. Eddie Cheng for their encouragement and insightful comments.

I am grateful to all my fellow graduate students.

Last but not least, I must express my very deep gratitude to my amazing parents, to my brothers and sisters, and to my best friends for the love, constant support and encouragement through my years of study and writing this thesis. This accomplishment would not have been possible without them. Thank you.

N.M

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Introduction

Among all the ideas generated by computer science over last 40 years, none has mutated the field as has parallel computation. Basically all aspects of computing were affected, and new concepts were achieved. From computer architecture to operating systems, from programming languages and compilers to databases and artificial intelligence, and from numerical to combinatorial computing, every branch of the discipline is undergoing a revival. In theoretical as well as in practical circles, a degree of movement is being experienced since the beginning of the computing era, the most noticeable effect was felt at the foundation of computer science, to be specific, *design and analysis of algorithms*. Just when the traditional study of algorithms was reaching to a stable state, the parallel computation revolution led to a new era of the field which is expected to continue for a long time.

Two reasons are usually given for studying this area: First, sequential computers (single processors) that perform one operation at a time, are quickly reaching a physical limit. This limit is imposed by the speed of light in a vacuum. Moreover, the time taken to obtain a solution is unacceptably slow. The sequence of instructions is the

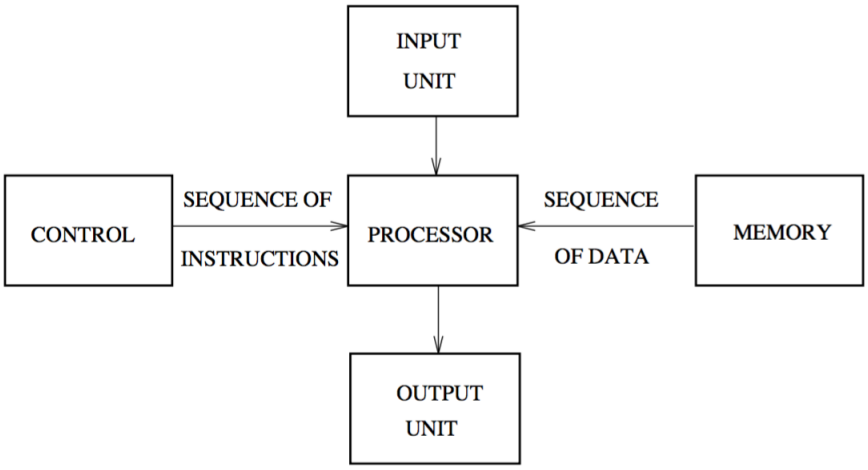

Figure 1.1: Sequential Computer

program, which causes the processor to reach the solution for a certain *problem*. The sequence of data is an *instance* of that problem. The control unit at each step brings out one instruction that operates on a datum (such as a pair of numbers) received from the memory unit. The instruction may be some arithmetic or logical operation on the datum and then the processor puts the result back into memory, considering that the processor has a small local memory, consisting of a constant number of fixed-size registers, to perform its computations. The processor is also connected to an input unit and output unit for communicating with the outside world. That is the procedure of computing the result in a *sequential* (or *serial*, or *conventional*) computer, Fig. 1.1.

A *parallel computer* solves a problem by utilizing several processors. A problem is broken into a number of subproblems. All of these subproblems are solved simultaneously, each on a different processor. In doing so, processors may communicate with one another to exchange partial results. Finally, the results are combined to produce an answer to the original problem.

The second reason for studying parallel computers is that they sometimes allow problems to be solved that otherwise are impossible to reach sequentially even without

considering taken time. There is however a third persuading reason for studying parallel computation: Conceptually employing several processors to work together on a given computation terminates in a totally new paradigm in computer problem solving. It offers precious techniques for the design and analysis of algorithms.

Parallel computers are classified into two major categories, namely, *shared-memory parallel machines* and *interconnection networks*, depending on how the processors communicate with each other.

In this chapter, we first introduce the classification of computer architectures. Then we explain two major computational models, shared-memory machines and interconnection networks, as well as introducing some well-known examples of the latter, since in this thesis the studies are done for two important interconnection networks, the (n, k)-*arrangement graphs* and *augmented cubes*. At the end of this chapter we discuss the parameters used to measure and analyze parallel algorithms in order to understand their efficiency. Finally, we give an overview and the organization of this thesis.

1.2 Classification of Computer Architecture

Different methods are proposed to characterize parallel computer architectures. In [13] computers are categorized into following four classifications according to the interaction between instruction streams and data streams.

1. **SISD - Single Instruction, Single Data Stream**

   This organization uses no parallelism in either instruction or data stream. So it is the most conventional computing equipment.

2. **SIMD - Single Instruction, Multiple Data Stream**

   This type of organization has multiple processors that are controlled by a central control unit. Each processor has its own memory unit. The same instruction

is applied by each processor on its data. All the major parallel models in this thesis are from this class of computers.

3. **MISD - Multiple Instruction, Single Data Stream**

   Here each processor has its own control unit and one common memory unit is shared among all processors. In these kind of computers a bunch of instructions are applied in parallel on the same data.

4. **MIMD - Multiple Instruction, Multiple Data Stream**

   This includes organizations known as "multiprocessor". Here, multiple processors execute different instructions on different data streams at the same time. This makes MIMD computer the most powerful ones among these four classifications.

SIMD and MIMD machines can be a member of the shared-memory family or of the interconnection network family.

1.3 Shared Memory Parallel Computers

Parallel random access machine (PRAM) is one model of parallel computation, shown in Fig.1.2. There exist N identical processors. In principle, N is an arbitrary large but finite number. These are connected to a single shared memory with M locations. M is unbounded in principle. The processors use this memory for their communication. A memory access unit (MAU) allows the processors to gain access to memory. Any pair of processors wishing to exchange data can do so through the shared memory. One processor writes its datum in a given memory location, which is then read by the other processors.

Four classifications based on different ways for the processors to gain access to memory for reading and writing are as follows:

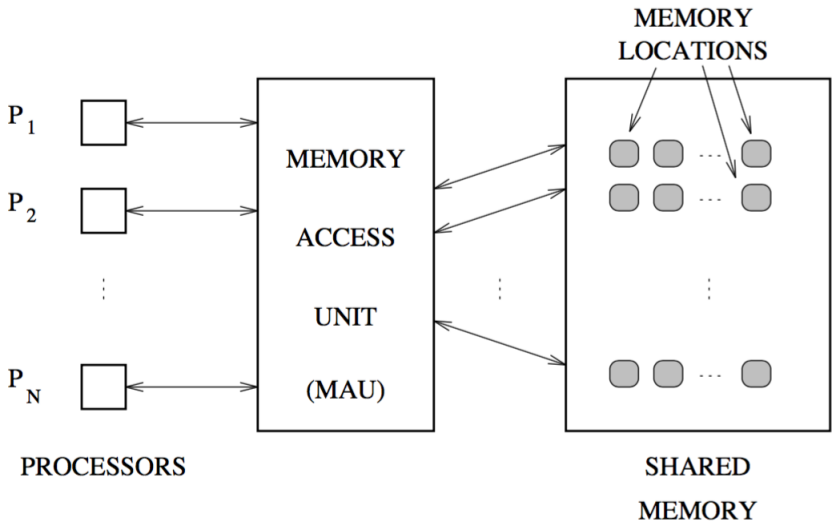

Figure 1.2: Parallel Random Access Machine

Exclusive Read(ER) In this form of memory access, just one processor can read from a memory location at a time, as shown in Fig1.3(a).

Concurrent Read(CR) In this form of memory access, two or more processors can read from the same memory location at the same time, as shown in Fig1.3(b).

Exclusive Write(EW) In this form of memory access, just a single processor can write into a memory location at a time, as shown in Fig1.4(a).

Concurrent Write(CW) In this form of memory access, two or more processors can write into the same memory location at the same time, as shown in Fig1.4(b).

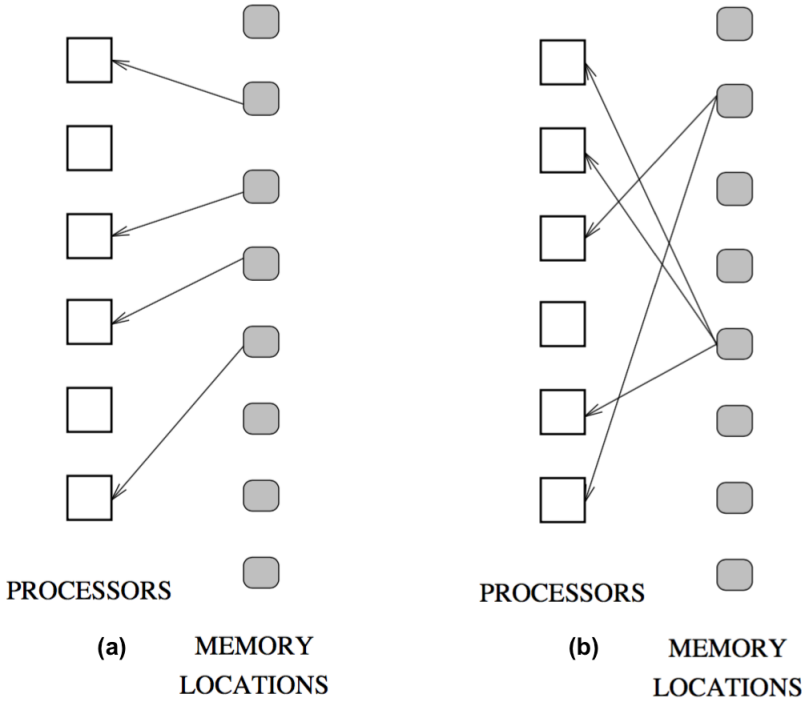

Figure 1.3: Read access to memory in PRAM: (a) Exclusive; (b) Concurrent.

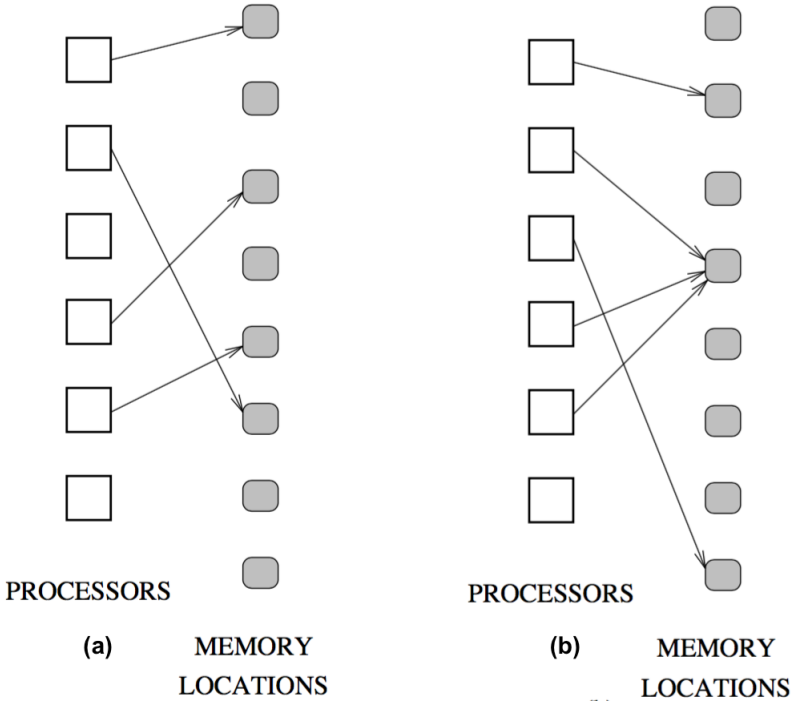

Figure 1.4: Write access to memory in PRAM: (a) Exclusive; (b) Concurrent.

By having these options, we can get four different types of PRAM computers as follows:

1. Exclusive Read, Exclusive Write (EREW)
2. Exclusive Read, Concurrent Write (ERCW)
3. Concurrent Read, Exclusive Write (CREW)
4. Concurrent Read, Concurrent Write (CRCW)

1.4 Interconnection Networks

Processors can also communicate directly through the links connecting them to each other. In this architecture there are M locations of memory distributed among N processors instead of one shared memory, Fig.1.5. Two processors that are connected by one link are called *neighbors*. If the link is a *two-way* communication link, data can be exchanged at the same time between these processors. Trees, meshes, hypercubes, and star graphs are popular interconnection networks.

Since interconnection networks are represented by undirected graphs $G = (V, E)$, we can use their properties and parameters to evaluate and analyze the network. In the following section we will review some definitions of graph theory terminologies that are going to be used in this thesis. We follow the definitions for graphs given in [11]. It should be mentioned that the terms "processor" and "node" and "vertex", "edge" and "link", "interconnection network" and "graph" will be used interchangeably.

1.4.1 Definitions

Definition 1. *A **directed graph** (or digraph) G is a pair (V, E), where V is a finite set and E is a binary relation on V. The set V is called the vertex set of G, and its elements are called vertices (or nodes). The set E is called the edge set of G and*

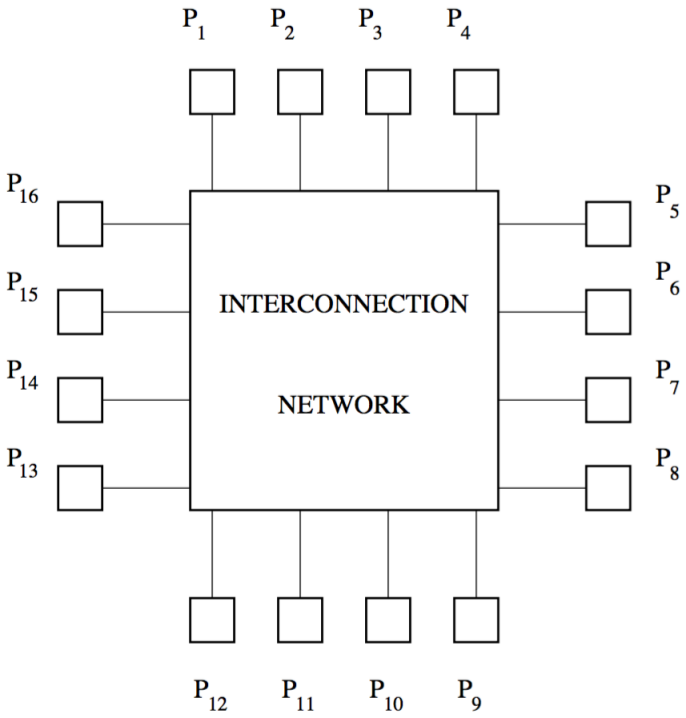

Figure 1.5: Interconnection network parallel computer.

its elements are called edges (links). An edge from node u in V to node v in V is represented as an ordered pair (u, v).

Definition 2. *In an **undirected graph** $G = (V, E)$, the edge set E consists of unordered pairs of vertices, rather than ordered pairs. Therefore, edges (u, v) and (v, u) are the same edge. In this case, we say that u and v are **neighbors**.*

For graph $G = (V, E)$, we sometimes use $V(G)$ to denote the vertex set of G and $E(G)$ to denote the edge set of G.

All graphs considered in this thesis are undirected.

Definition 3. *The **degree** of node $v \in V$ in a graph G, is the number of neighbors of v. The degree of a graph is said to be the maximum of all node degrees.*

Definition 4. *A graph $G = (V, E)$ is called **regular** when all the nodes in G have the same degree. When n is the degree of each node (the degree of the graph), we call the graph n-regular.*

Definition 5. *A **path** of length k between nodes u and v in a graph $G = (V, E)$ is a sequence $< u_0, u_1, ..., u_k >$ of vertices such that $u = u_0, v = u_k$, and $(u_{i-1}, u_i) \in E$ for $i = 1, 2, ..., k$. The length of the path is the number of edges in the path. A path is simple if all vertices in the path are distinct.*

All paths discussed in the thesis are simple paths.

Definition 6. *The **distance** between node u and node v is the length of a shortest path between u and v.*

Definition 7. *The **diameter** of the graph is the maximum distance between any two nodes $u, v \in V$ of a graph G.*

A network with a small diameter is more desirable than one with a large one, since the time for passing on a message between two processors depends on the distance between them in a graph [2].

Definition 8. *An **isomorphism** from a graph G to a graph H is a bijection $f : V(G) \to V(H)$ such that $(u, v) \in E(G)$ if and only if $(f(u), f(v)) \in E(H)$. G is isomorphic to H if there is an isomorphism from G to H. An **automorphism** of a graph is an isomorphism from G to G.*

Definition 9. *A graph is **node-symmetric (edge-symmetric)** if for every pair of nodes $u, v \in V$ (edges $e, f \in E(G)$), there is an automorphism that maps u to v (e to f).*

This means that in a symmetric graph, if you look at the whole graph from any node (edge) of $V(G)$ ($E(G)$) it will be exactly the same graph.

One important consequence of the vertex symmetry is that a structure embedded in one part of the network can be easily transformed into another part of the network without loosing its efficiency. Also routing and broad casting algorithms can be designed much easier due to same accessibility between nodes in these kind of graphs.

This characteristic is used in proposed algorithms in this thesis. A special class of graphs, namely the *Cayley Graphs* are all vertex and edge symmetric [1], [19].

Definition 10. *A graph G is called f- **fault tolerant** when the removal of f or less nodes does not make the graph disconnected. The fault tolerance of a graph G is the largest value for f that G is f-fault tolerant.*

This property indicates the maximum number of faulty or blocked nodes that can exist in a graph, but the network is still able to continue.

Definition 11. *A **bipartite** graph is a graph whose nodes can be divided into two distinct sets V_1 and V_2 in a way that there is no edge between the nodes of each set. Each edge connects a node from one set, V_1 to a node from the other set, V_2.*

There exist many interconnection networks, each performing better than the other ones under specific conditions. We will review some well known interconnection networks in the rest of this section.

1.4.2 Complete Graph

One way for N processors to be connected is to use two-way link for connecting each pair of processors, Fig.1.6 shows such a graph which is called a *complete network*. Therefore, the numbers of links in this network are $N(N-1)/2$ making it impossible in practice since the number of links that can be physically connected to a processor is limited. Also implementation of such a network would be so expensive because of the total number of links. So we are more interested in more reasonable graphs.

1.4.3 Linear Array and Ring

A linear array is a simple topology. Here N processors are interconnected in a one dimensional array. So each processor has 2 neighbors, except the end processors which

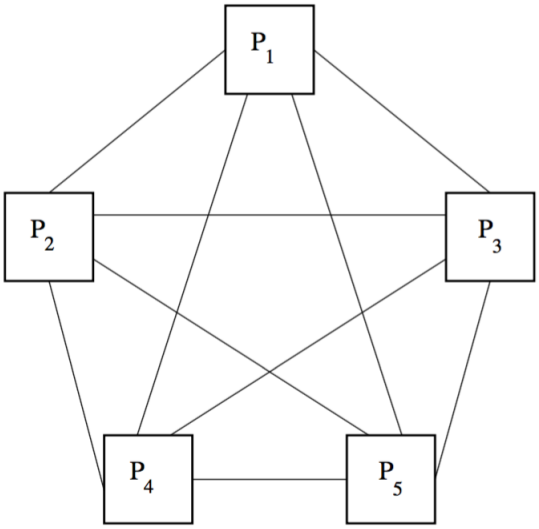

Figure 1.6: A complete network for $N = 5$

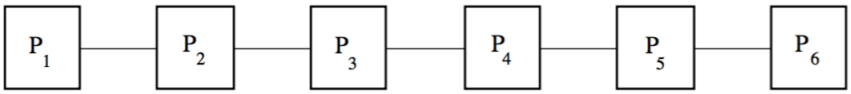

Figure 1.7: Linear array interconnection network.

have only one neighbor. A linear array is shown in Fig.1.7. A *ring* of processors is a special case of linear array where the first and the last one are connected to each other.

1.4.4 Mesh and Torus

The corresponding undirected graph of the mesh interconnection network is a two-dimensional array. The neighbors of processor $P_{i,j}$, which is located in row i and column j, will be $P_{i-1,j}$, $P_{i+1,j}$, $P_{i,j-1}$ and $P_{i,j+1}$, except for the processors on the boundary rows and columns. Each processor has 4 neighbors expect the ones on the boundary rows and columns. Thus the degree of mesh is 4. Fig.1.8 shows a 4×4 mesh, the row and column indices are shown in the graph. A *Torus* is a special case of mesh where the first and the last one in each row and column are connected to

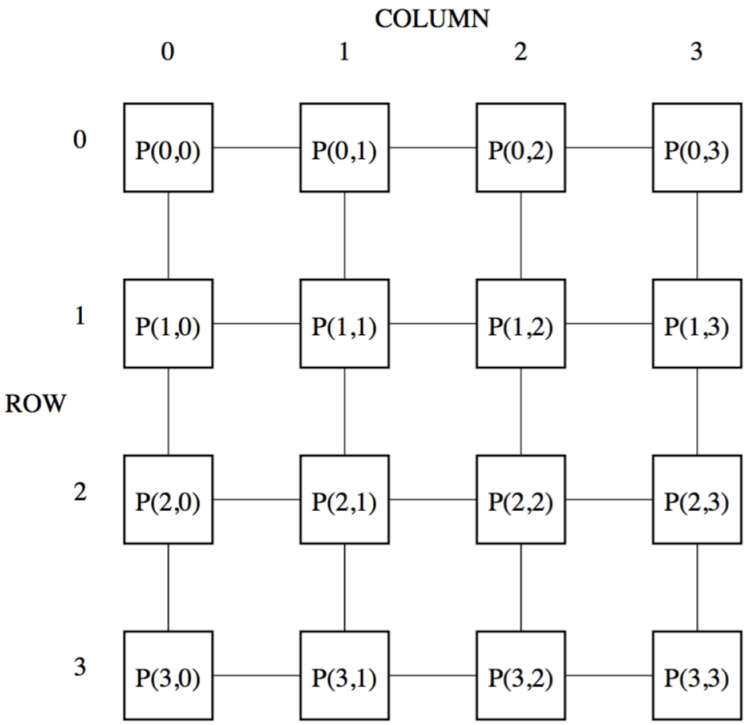

Figure 1.8: Mesh interconnection network

each other.

1.4.5 Hypercube

An n dimensional hypercube, Q_n, also known as n-cube graph is an undirected graph consisting of $K = 2^n$ vertices labeled from 0 to $2^n - 1$ and such that there is an edge between any two vertices if and only if the binary representations of their labels differ by one and only one bit. Thus, each node has n links connected to n neighbors.

The first important property of the n-cube is that it can be constructed recursively from lower dimensional cubes. The *Hamming distance*, or minimum distance, between to nodes p and q is equal to the number of different bits in their binary representations. It is straight forward to see that the diameter of the n-cube is n. Other topologies such as rings, meshes and trees can be mapped into hypercube. It also has some other attractive properties, including regularity, node symmetric, edge symmetric,

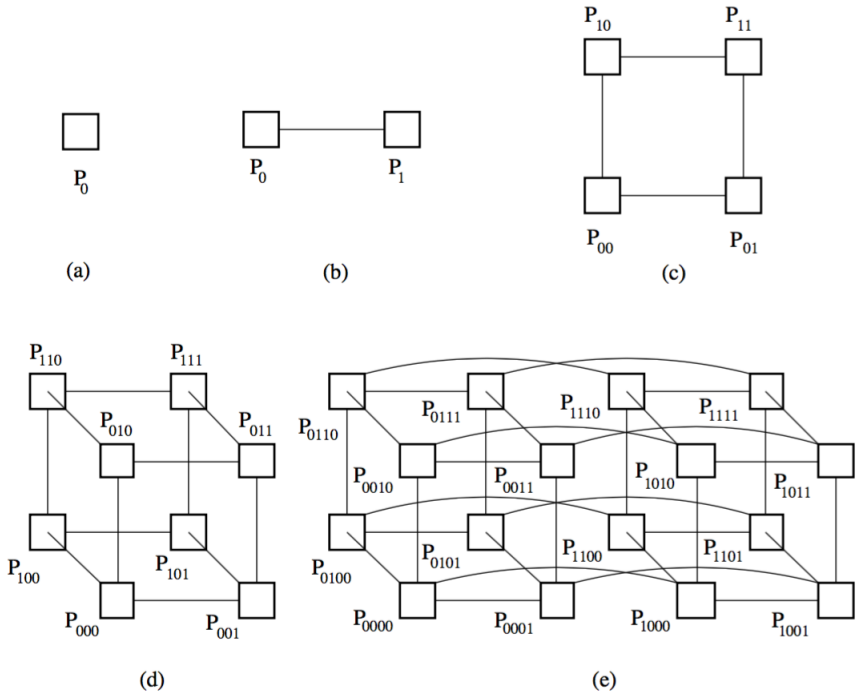

Figure 1.9: (a) 0-cube; (b) 1-cube; (c) 2-cube; (d) 3-cube; (a) 4-cube

small diameter and strong connectivity [18]. Fig.1.9 shows n-cubes for $n = 0, 1, 2, 3$ and 4. It shows that a 4-cube is made up of two disjoint 3-cubes, i.e. a left 3-cube (consisting of eight nodes whose first bits are all 0's) and a right 3-cube (consisting of eight nodes whose first bits are all 1's). These two 3-cubes are connected by eight cross links. The operation of splitting an n-cube into two $(n-1)$-cube such that their nodes are in a one-to-one correspondence is called *tearing*. Tearing an n-cube into two $(n-1)$-subcubes can be done in n different ways [27].

Node-to-Node Communication in the Hypercube

A functional routing algorithm for sending a message from a source node to a destination node in the hypercube is presented in [29] by Tzeng and Wei. This routing algorithm always finds the shortest paths for sending messages. During routing a *tag* is also carried along with a message that is the *relative address* of the *current* node and destination node. The relative address of two nodes is the bitwise Exclusive-OR

of their addresses, $p \oplus q$. Also in this algorithm numbers are assigned to the edges. An edge connecting two nodes that are different in the i-th bit position is considered to have link number i (starting from left to right consider the first bit as bit 1). For example, the link between nodes 1010 and 1110 has number 2. Here we describe a procedure for transmitting a message between two nodes in the hypercube. This is going to be used further in this thesis for designing an algorithm for augmented cubes.

Algorithm 1 Routing Algorithm for Q_n

Finding a shortest path from a source node, s, to a destination node, t;

•$tag \longleftarrow s \oplus t$

1. **If** $tag = 0$
   - STOP; the message has reached t
2. **Else**
   - Starting from left bit to right one of tag:
   - Let i be the bit number of the first 1 in tag
   - Send the message on link i and set bit i in tag to zero

1.4.6 Cube Connected Cycles

Despite being a popular architecture, the hypercube has one major disadvantage that the node degree grows linearly with the hypercube dimension. Consequently, the hypercube is not an appropriate choice for very large scale integration, VLSI implementation. Preparata and Vuillemin [25] introduced the cube-connected cycle (CCC) architecture as a substitutive architecture.

A CCC is obtained by replacing each node in an n-cube by a cycle of n nodes. Each node will be labeled with a pair (i, w), where w is an n-bit binary address that denotes the cycle of the node and i is the dimension of the node ($0 \leq i \leq n-1$). Two

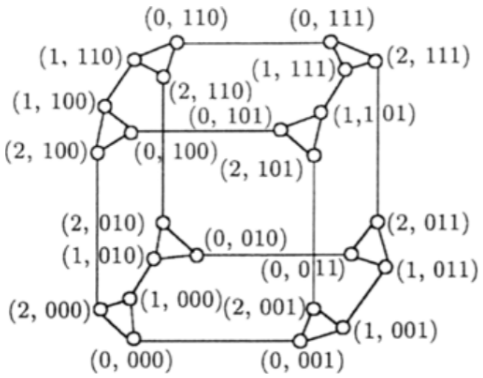

Figure 1.10: $3D$ cube connected cycle

nodes (i, w) and (i', w') are linked by an edge in the CCC if and only if either:

1. $w = w'$ and $i - i' \equiv \pm 1 \; mod \; n$, or

2. $i = i'$ and w differs from w' in precisely the i-th bit.

As an example a 3-dimensional cube connected cycle (3-CCC) is shown in Fig.1.10.

Edges of the first type are called *cycle edges* in cycle w, while edges of the second type are referred to as *hypercube edges* in dimension i [28].

The CCC not only preserves all the attractive features of the hypercube, such as small diameter and symmetry, but also has fewer links and constant node degree 3, making it ideal for VLSI implementation [25].

1.4.7 Star Graph

In a star graph, S_n, of dimension n, there are $n!$ nodes where each node is a permutation of symbols $1, 2, ..., n$. Two permutations are connected if one can be obtained from the other by swapping its symbols at positions 1 and i, $2 \leq i \leq n$.

S_n can be decomposed into n sub-stars S_{n-1}, namely, $S_{n-1}(i)$, $1 \leq i \leq n$ where all the vertices have i for their last symbol in their binary representation [2]. $S_{n-1}(i)$ is an $(n-1)$-star defined on symbols in $\{1, 2, ..., n\} - \{i\}$. S_4 in Fig.1.11 contains four 3-stars, by fixing $1, 2, 3, 4$ as their last symbols.

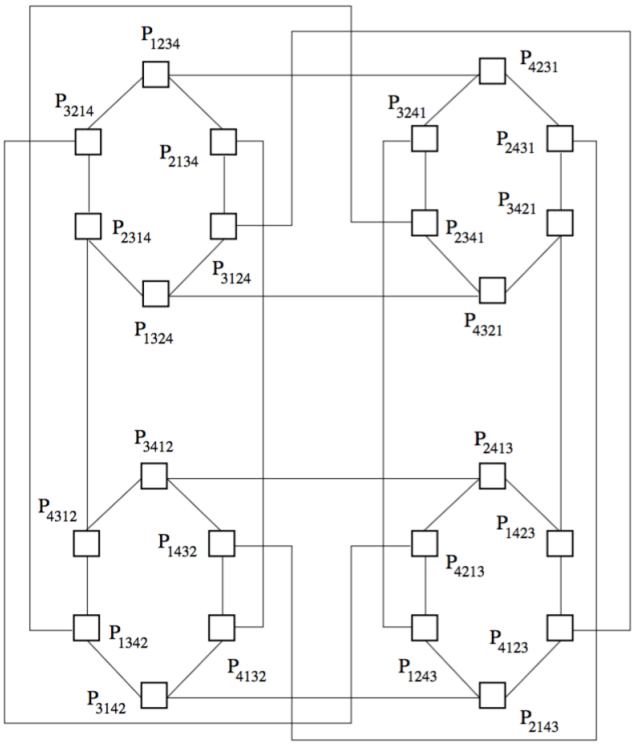

Figure 1.11: 4-star interconnection network

The star graph is an attractive alternative to the hypercube and compares favorably with it in several aspects. Also, it is known that the star graph is both vertex-symmetric and edge-symmetric [7].

Node-to-Node Disjoint Paths in the Star graph

In [26] an $O(n^2)$-time algorithm is presented for finding $n-1$ disjoint paths in the star interconnection network with path lengths no more than $d(s,t)+4$, where $d(s,t)$ is the shortest path length between node s and node t. Each one of $n-1$ disjoint paths is located in a different sub-star which proves that these paths are certainly disjoint. Also by using a simple routing within each sub-star the proof of path lengths is also presented in [26]. In the following procedure, routing is started from an arbitrary node, labeled by source node, to reach to the identity node, which is called destination. This procedure is described bellow since it is used in this thesis.

Algorithm 2 Routing Algorithm for S_n

Finding $n-1$ node disjoint paths from a source node, s, to a destination node, t;

1. Routing source node to its $n-1$ neighbors, in no more than two steps
   - These nodes are located in $n-1$ sub-star $S_{n-1}(i)$
2. Routing each of $n-1$ neighbors, $u(i)$, to $n-1$ nodes with distance 2 from destination through a shortest path following rules:
   - **If** $u_1(i) = 1$, move it to any position not occupied by the correct symbol
   - **If** $u_1(i) = x \neq 1$, move it to its correct position
3. Routing each of $n-1$ nodes to the identity node, in two steps

There are many other interconnection networks such as the *hyperstar* [20], [3], [30], the (n,k)-*star graph* [9], the *alternating group graphs* [17], [5], [4], [23], (n,k)-*arrangement graphs*, and *augmented cubes*.

1.4.8 Arrangement Graph

As mentioned earlier the star graph is an attractive alternative to the n-cube with noteworthy advantages such as: a lower degree and a smaller diameter. However, the star graph has one major weakness corresponding to its numbers of nodes. For instance the smallest star graph with at least $8K$ nodes is an 8-star graph with roughly $40K$ nodes. Consequently, because of the large gap between $n!$ and $(n+1)!$, one may face the choice of either too few or too many available nodes [9]. Day and Tripiathi [12] proposed the arrangement graph as a solution for this difficulty. This interconnection topology contains star graph and holds on all the nice characteristics of the star graph such as node and edge symmetry, hierarchical structure and simple shortest path routing. An (n,k)-arrangement graph, $A_{n,k}$ $(1 \leq k \leq n-1)$, is a graph

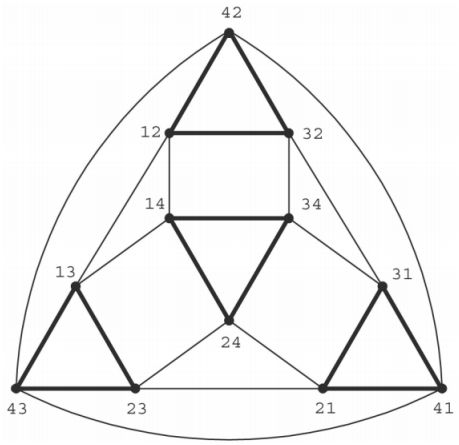

Figure 1.12: $(4, 2)$-arrangement graph

with all the k-permutations of n as vertices where two k-permutations are adjacent if they agree in exactly $k-1$ positions. It is a regular graph of degree $k(n-k)$ with $n!/(n-k)!$ number of nodes and diameter $\lfloor \frac{3}{2}k \rfloor$. Fig1.12 illustrates an $A_{4,2}$.

The $(n, n-1)$-arrangement graph is isomorphic to the n-star, and the $(n, 1)$-arrangement graph is isomorphic to the complete graph with n nodes. The number of nodes in an arrangement graphs gives us more choices in the number of nodes other than the number of the nodes in the star graph. For example an arrangement graph of with at least $8K$ nodes can be obtained for each of the combinations $(n = 8, k = 6)$ and $(n = 9, k = 5)$. The first combination yields a degree of 12 and a diameter of 9 , and the second combination yields a degree of 20 and a diameter of 7.

1.4.9 Augmented Cube

Many different variations of hypercubes have been studied to enhance the efficiency of hypercubes. Chudum and Sunitha [10], used two permutations of vertex set of n-dimensional hypercube, to join a vertex from one copy of the graph, to the vertices of the other copy, with the same dimension. The result is a study on augmented cube. They have proposed different definitions for augmented cube, here we present

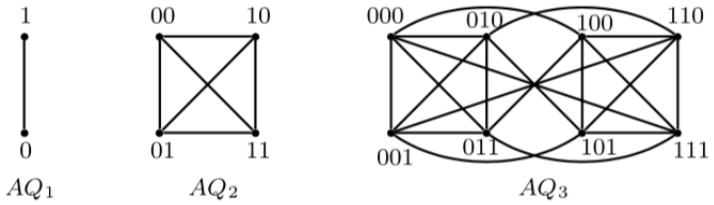

Figure 1.13: Three augmented cubes AQ_1, AQ_2 and AQ_3

a simple one.

The augmented cube, AQ_n, of dimension n, has 2^n nodes. These vertices are labeled with n-bit binary strings. Two nodes, $p = p_1p_2p_3...p_n$ and $q = q_1q_2q_3...q_n$, are connected if and only if there exists an integer l, $1 \leq l \leq n$ such that either:

1. Their binary representations only differ in bit l

2. Their binary representations are exactly the same for the first $l-1$ bits, and for $l \leq i \leq n$ the binary representation of one node is the complement of the other one.

The graphs shown in Fig.1.13 are AQ_1, AQ_2 and AQ_3, respectively.

Since the main focus on this thesis is on the (n,k)-*arrangement graphs* and *augmented cubes*, their characteristics and related algorithms are discussed in detail further in this thesis.

Table 1.1 lists topologies we described earlier and some of their important properties.

1.5 Evaluating Parallel Algorithms

Different factors are considered to evaluate an algorithm as a good one. The most important ones are the algorithm's running time, number of processors, and its cost.

For measuring the speed of an algorithm the total number of steps executed by an algorithm is used. We will always consider the worst case for problems, as in this

Table 1.1: Interconnection networks and some of their parameters

Network	Number of Nodes	Degree	Diameter
Complete Graph	n	$n-1$	1
Linear Array	n	2	$n-1$
Ring	n	2	$\lfloor n/2 \rfloor$
Mesh $(m * n)$	mn	4	$(m-1)+(n-1)$
Torus $(m * n)$	mn	4	$\lfloor (m+n)/2 \rfloor$
Hypercube	2^n	n	n
Cube Connected Cycles	$n2^n$	3	$2n + \lfloor n/2 \rfloor - 2$ for $n \geq 4$
Star Graph	$n!$	$n-1$	$\lfloor 3(n-1)/2 \rfloor$
Arrangement Graph	$n!/(n-k)!$	$k(n-k)$	$\lfloor \frac{3}{2}k \rfloor$
Augmented Cube Graph	2^n	$2n-1$	$\lceil n/2 \rceil$

case we will have the maximum number of steps. A parallel algorithm usually has two types of elementary steps:

1. *Computational step*: which is a basic arithmetic or logical operation performed on one or two data, such as adding and comparing.

2. *Routing step*: which is moving a datum of constant size from one processor to another. This replacement will occur through shared memory or a direct link between these two processors.

Generally, computational step require a constant number of time units, while replacing a datum depends on the distance between the processors.

1.5.1 Running Time

This is the time starting from when the first processor begins doing the first operation till the last processor finishes producing the result. The running time is measured by counting the number of elementary steps done by an algorithm to solve the most difficult case of a problem. For a problem of size n, the worst case running time of parallel algorithm would be represented by $t(n)$.

1.5.2 Number of Processors

Several reasons are given in [2] for considering this number in analysis of an algorithm. Briefly, we are interested in systems with fewer processors, in a case that the running time and type of parallel computer used, are the same. The reason is that the more processors used, the more expensive computers will be. For a problem of size n, the number of processors needed by an algorithm would be represented by $p(n)$.

1.5.3 Cost

The cost of an algorithm, $c(n)$, is the product of time consumed in the worst case and the number of processors it uses. That is, $c(n) = t(n) \times p(n)$.

The factor to find out if a parallel algorithm is beneficial to use, comparing with the sequential algorithms is called ***efficiency***. Consider t_1 as the worst case running time of the best and fastest sequential algorithm that exists for a given problem. The cost of the parallel algorithm for solving the same problem would be $p(n) \times t(n)$, and its efficiency is:

$$E(1,p) = \frac{t_1}{p(n) \times t(n)}$$

We have following cases for evaluating the efficiency:

1. If $E(1,p) < 1$, then the parallel algorithm's cost is not optimal.
2. If $E(1,p) = 1$, then the parallel algorithm's cost is optimal.
3. If $E(1,p) > 1$, then there is a faster sequential algorithm by simulating the parallel one.

1.6 Organization of the Thesis

Different interconnection networks have been proposed before and there are still new networks being proposed. These new networks are mostly the improved form of the

previous ones. For example, the augmented cube is one of the improved versions of the hypercube network.

There are different routing paradigms for networks. In this thesis, our focus is on the bounded disjoint paths paradigms. For example, how should we find the maximum number of disjoint paths between a source node and a set of destination nodes such that no two paths share a common node except the source? Under which conditions are those disjoint paths bounded?

This thesis proposes a proof that there are disjoint paths between one source node and $k(n-k)$ target nodes in the (n,k)-arrangement graph. Also an algorithm is presented for finding $2n-1$ disjoint shortest paths in an augmented cube. Chapter 2 and Chapter 3 provide literature reviews of these two graphs. In Chapter 4, we prove that there are $k(n-k)$ disjoint paths between the source node and $k(n-k)$ target nodes. The lengths of paths are no more than $diameter + (n-k) + 2$. Also we present an optimal routing for finding n disjoint paths between 2 nodes in an (n,k)-arrangement graph. Our algorithm runs in $O(n^2)$ time. In Chapter 5 we study the problem of finding shortest and disjoint paths between a source node and $2n-1$ target nodes in an augmented cube. Our algorithm checks to see whether such paths exist and in case they do, finds them, all in $O(n^3)$ time. Chapter 6 is dedicated to conclusion and future work.

Chapter 2

Literature Review

Arrangement Graph

2.1 Introduction

The (n, k)-arrangement graph is proposed in 1992 [12] as a generalization of the star graph topology. The (n, k)-arrangement graph has attracted lots of attentions as a new topology of interconnection networks. The n-star has many appealing properties; however, its major problem is a huge gap between the number of nodes in an n-star and an $(n + 1)$-star, i.e. there is a huge difference between $n!$ and $(n + 1)!$ [22]. This is the main reason why (n, k)-star graph [9] and (n, k)-arrangement graph [12] are proposed, both as a generalization of the star graph. Compared with the n-star, the (n, k)-arrangement graph is more flexible in degree and diameter.

In this chapter we provide a literature review of the (n, k)-arrangement graph and a review on its already proven properties.

2.2 Properties

Consider two integers n and k such that $1 \leq k \leq n-1$. Let $[n] = \{1, 2, ..., n\}$. A k-permutation is $p_1p_2...p_k$ where $p_i \in [n]$.

Definition 12. *The arrangement graph, $A_{n,k} = (V, E)$ is an undirected graph with all the k-permutations of $[n]$ as vertices. Two k-permutations $p = p_1p_2...p_k$ and $q = q_1q_2...q_k$ are directly connected by an edge to each other if they agree in exactly $k-1$ positions. That is:*

$V = \{p_1p_2...p_k | p_i \in [n]$ *and* $p_i \neq p_j$ *for* $i \neq j\}$

and

$E = \{(p, q) | p, q \in V$ *and for an* $i \in [k], p_i \neq q_i$ *and* $p_j = q_j$ *for all* $j \neq i\}$.

Definition 13. *The connecting edge between the two permutations that differ in position i is called i-edge. Here, p and q are called i-adjacent.*

Given a vertex $p \in V$, a vertex q obtained by swapping two symbols of p, cannot be its neighbor since they are different in two positions. Thus, every neighbor for a node is obtained by replacing one of its elements by an element other than any of its elements [6].

For example in $A_{6,3}$ the node $p = 235$ is connected to the nodes 135, 215, 231, 435, 245, 234, 635, 265, and 236.

Proposition 1. *$A_{n,k}$ is a regular graph with degree $k(n-k)$ and $n!/(n-k)!$ nodes [12].*

Theorem 1. *$A_{n,k}$ is vertex symmetric and edge symmetric [12].*

Proof is given in [12].

Vertex symmetry in arrangement graphs simplifies the problem of routing between two arbitrary nodes p and q to the problem of routing between an arbitrary node p and the special identity node $I_k = 12...k$ [21].

2.2.1 Cycle Structure for Permutation

Here we are going to define a cycle representation for each node. A permutation of n symbols can be viewed as a set of cycles in a way that each symbol's desired position is occupied by the next symbol. This cycle representation will be used in routing algorithms.

Suppose that we have a permutation $p = p_1p_2...p_k$. The set EXT$(p) = [n] - \{p_1, p_2, ..., p_k\}$ is the $n-k$ *external elements* of p from $[n]$ which are not used in the permutation p. Also the set INT$(p) = \{p_1, p_2, ..., p_k\}$ is the k *internal elements* of p from $[n]$ that are used in the permutation p. *Identity* node is a special node $12...k$ denoted by I_k. External elements of identity node are called *foreign elements*. We can always represent the nodes of an arrangement graph using a set of internal and external cycles which contain only misplaced elements and should be of length at least two. Also each external cycle contains exactly one foreign element. In the external cycle, the first element will be a non-foreign element which does not appear in the permutation p while the foreign element that appears in p will be written as the last element [12].

Example. Consider the node $p = 74253$ of the $A_{8,5}$ arrangement graph. The cycle representation for this node will be $C_1 = (2, 4, 5, 3)$ and $C_2 = (1, 7)$. Cycle C_1 is an internal cycle since all its elements are internal, but cycle C_2 is an external one because of element 7 which is a foreign element. First cycle means that in this permutation position 2 is held by 4, position 4 is held by 5, position 5 is held by 3, and position 3 is held by 2. In second cycle position 1 is held by 7.

2.2.2 Routing and Shortest Length Path

As mentioned before, because of the vertex symmetry in arrangement graphs, routing between any two nodes can be mapped to routing from an arbitrary node p to the identity node. In this case, routing can be done by first writing cycle representation

for node p and then correcting them one by one. Correcting a cycle means placing each of its elements in its correct position [21]. By having the external cycle in a way described earlier, it can be corrected by first moving its non-foreign external element to its correct position held by next element, this procedure will be repeated until the last element, a foreign element, is made external when it is replaced by the previous element. Correction of an internal cycle requires to exchange the first element of the cycle with an arbitrary external element, x. So, the first element is now an external one and can be moved to its correct position which is occupied by the next element of the cycle. We will continue this procedure until the last element of the cycle is taken to its correct position making x external element again.

Example. Following the above procedure, the external cycle $C_2 = (1, 7)$ in $A_{8,5}$ of the node $p = 74253$ is corrected as bellow:

$74253 \rightarrow \underline{1}4253$,

while the internal cycle $C_1 = (2, 4, 5, 3)$ of the node $p\prime = 14253$ is corrected along the path:

$14253 \rightarrow 14\underline{6}53 \rightarrow 1\underline{2}653 \rightarrow 126\underline{4}3 \rightarrow 1264\underline{5} \rightarrow 12\underline{3}45$.

Each underlined element shows the action at each step.

According to the above routing algorithm, an upper bound for distance between an arbitrary node and the identity node is derived in [12]. Also it is shown that the length of this path is minimum. Let $C_1, ..., C_e, C_{e+1}, ..., C_t$ be a cycle structure for a node $p = p_1p_2...p_k$ in $A_{n,k}$, such that $C_1, ..., C_e$ are external cycles and the rest are internal cycles. m, t, and e are defined as follows:

- m: total number of elements in these cycles
- t: total number of cycles
- e: number of external cycles

Lemma 1. *The distance $d(p, I_k)$ between p and I_k in $A_{n,k}$ satisfies $d(p, I_k) \leq t+m-2e$*

[12].

Proof. For correcting an external cycle with m_i elements, $m_i - 1$ steps are required. So for correcting all the external cycles we need to do $(m_1-1)+(m_2-1)+...+(m_e-1) = (m_1 + m_2 + ... + m_e) - e$ steps. Also number of steps for correcting an internal cycle with m_j elements would be $m_j + 1$. Correction all the internal cycles needs $(m_{e+1}+1)+(m_{e+2}+1)+...+(m_t+1) = (m_{e+1}+m_{e+2}+...+m_t)+(t-e)$. Therefore,
$d(p, I_k) \leq m_1 + m_2 + ... + m_e - e + m_{e+1} + m_{e+2} + ... + m_t + t - e$
$= t + m - 2e$, [12]. □

Theorem 2. *The distance between p and I_k in $A_{n,k}$ is $d(p, I_k) = t + m - 2e$ [12].*

The proof is given in [12].

Corollary 1. *The diameter of $A_{n,k}$ is $\lfloor \frac{3}{2}k \rfloor$ [12].*

Proof. The proof given in [12] is as follows. By *Theorem 2* the maximum distance in an $A_{n,k}$ is $d(p, I_k) = t+m-2e$. This equation can have its maximum value when there is no foreign element in the cyclic representation of a node. So there is no external cycle, $e = 0$. Also every two elements form an internal cycle, which means $m = k$ and $t = \lfloor k/2 \rfloor$; therefore the diameter is $\lfloor \frac{3}{2}k \rfloor$. □

2.2.3 Hierarchical Structure

We know that there are $n!/(n-k)!$ nodes in an $A_{n,k}$ graph. If element i is fixed in position j, for i and j, $1 \leq i \leq n, 1 \leq j \leq k$, an $A_{n-1,k-1}$ sub-graph is formed with $(n-1)!/(n-k)!$ number of nodes. This sub-graph of $A_{n,k}$ consisting of all nodes with element i in position j is denoted by i_j. The sub-graphs $1_j, 2_j, ..., n_j$ form n disjoint sets of vertices of $A_{n,k}$. This partitioning into n copies of $A_{n-1,k-1}$ can be done in k different ways.

Also the partitioning can be done by fixing one element i to be in any k positions to get $i_1, i_2, ..., i_k$, and a set, i_0, of all nodes that do not have i. Here we have k copies

of $A_{n-1,k-1}$ and one copy of $A_{n-1,k}$. This partitioning can be done in n different ways $(1 \leq i \leq n)$ [12].

Theorem 3. *$A_{n,k}$ contains $\binom{k}{a}\frac{n!}{(n-a)!}$ copies of $A_{n-a,k-a}$, for $1 \leq a \leq k-1$ [12].*

Proof. A copy of $A_{n-a,k-a}$ is obtained by fixing a elements out of k elements and changing remained $k-a$ elements. For selecting a positions out of k positions, there exist $\binom{k}{a}$ different ways. Also there are $n!/(n-a)!$ ways for selecting a elements out of n available elements to assign to these positions. Here order of elements should be considered. Therefore, there are $\binom{k}{a}\frac{n!}{(n-a)!}$ copies of $A_{n-a,k-a}$ in $A_{n,k}$ [12]. □

This property of an arrangement graph is going to be used in the proposed algorithm for finding disjoint paths, while routing from an arbitrary node to I_k, later in this thesis.

Chapter 3

Literature Review

Augmented Cubes

3.1 Introduction

It is well known that the hypercube is one of the most popular interconnection networks because of its noteworthy properties such as maximum connectivity and effective routing algorithms.

The *augmented cube,* AQ_n, is proposed by Choudum and Sunitha [10] as an improvement over the hypercube Q_n, which owns more desirable properties than the hypercube, besides keeping some beneficial properties of Q_n. We know that Q_n contains cycles with even length, while AQ_n includes cycles of all lengths from 3 to 2^n. In [24] it has been proved that in AQ_n, between any two distinct vertices, there exist paths of all length from their distance to $2^n - 1$.

In this chapter some properties of the augmented cube are reviewed.

3.2 Properties

Let n be an integer such that $n \geq 1$. The degree of a vertex v is denoted by $deg(v)$.

3.2.1 Hierarchical structure

For $n \geq 2$, AQ_n can be recursively constructed by two copies of AQ_{n-1} which are connected by 2^n edges. These two copies of a lower dimension are divided based on their first bits of their binary representations. They are denoted by AQ^0_{n-1} and AQ^1_{n-1}, such that in AQ^0_{n-1}, all nodes' binary representations start with 0 and in AQ^1_{n-1}, all all nodes' binary representations start with 1.

Definition 14. *Let* $V(AQ^0_{n-1}) = \{0p_2p_3...p_n | p_i = 0 \text{ or } 1 \text{ for } 2 \leq i \leq n\}$ *and* $V(AQ^1_{n-1}) = \{1q_2q_3...q_n | q_i = 0 \text{ or } 1 \text{ for } 2 \leq i \leq n\}$. *A vertex* $p = 0p_2p_3...p_n$ *is connected to a vertex* $q = 1q_2q_3...q_n$ *if and only if either:*

1. $p_i = q_i$ *for every* i, $2 \leq i \leq n$; (p, q) *is called a hypercube edge or*

2. $p_i = \bar{q}_i$ *for every* i, $2 \leq i \leq n$; (p, q) *is called a complement edge.*

It is obvious that for $n \geq 2$, $deg(\alpha p) = deg(p) + 2$, for every vertex $p \in AQ_{n-1}$ and $\alpha \in \{0, 1\}$.

Proposition 2. *By having the recurrence relation for the degree of each node in* AQ_n, *every vertex,* $\alpha p \in AQ_n$ *has degree of* $2n - 1$. *So* AQ_n *is* $(2n - 1)$*-regular and has* $(2n - 1)2^{n-1}$ *edges [10].*

Proposition 3. AQ_n *has the following adjacency, properties proposed in [10]:*

1. *Two vertices* p *and* q *are adjacent if and only if* $\bar{p}$ *and* $\bar{q}$ *are adjacent.*

2. *If* $p \in AQ^0_{n-1}$ *(or* AQ^1_{n-1} *) is connected to* q *and* z *which are in* AQ^1_{n-1} *(respectively,* AQ^0_{n-1}*), then* q *and* z *are also connected to each other.*

Theorem 4. *For every* $n \geq 1$, AQ_n *is vertex symmetric [10].*

3.2.2 Path and Distance

Consider two vertices $p, q \in AQ_n$ and suppose that $p = \alpha x$, $q = \alpha y$, where $x, y \in AQ_{n-1}$ and $\alpha \in \{0, 1\}$. If $P(x, y)$ is an (x, y)-path in AQ_{n-1}, then by adding α to each of the vertices in this path, we have a (p, q)-path in AQ^{α}_{n-1}, indicated by $P^{\alpha}(\alpha x, \alpha y)$ [10].

Proposition 4. *Let $p, q \in AQ_n$, following properties are given in [10]:*

1. *If there is a path $P(p, q) : (p =)a_1, a_2, ..., a_t(= q)$, then we have a path $\bar{P}(\bar{p}, \bar{q}) : (\bar{p} =)\bar{a}_1, \bar{a}_2, ..., \bar{a}_t(= \bar{q})$.*

2. *$P(p, q)$ is a shortest path between p and q, if and only if $\bar{P}(\bar{p}, \bar{q})$ is a shortest path for $\bar{p}$ and $\bar{q}$,.*

3. *$d(p, q) = d(\bar{p}, \bar{q})$.*

Proof. The first item follows from *Proposition* 3, that if two nodes are adjacent, so are their complements. The second and third items follow from the first one. □

Theorem 5. *Let $p, q \in AQ_n$, following properties are given in [10]:*

1. *If both p and q are located in the same sub-graph, AQ^0_{n-1} (or AQ^1_{n-1}), then there exists a shortest path between these two nodes in AQ_n with all its vertices in the same sub-graph, AQ^0_{n-1} (respectively AQ^1_{n-1}).*

2. *If p and q are located in different sub-graphs, $p \in AQ^0_{n-1}$ and $q \in AQ^1_{n-1}$, then*

   (i) *there exists a shortest path between these two nodes in AQ_n with all its vertices except p in AQ^1_{n-1}. The second vertex of this path (the neighbor of p) is either $1p_2p_3...p_n$ or $1\bar{p}_2\bar{p}_3...\bar{p}_n$, based on which one has the smaller distance to p.*

(ii) there exists a shortest path between these two nodes in AQ_n with all its vertices except q in AQ^0_{n-1}. The second vertex of this path (the neighbor of q) is either $0q_2q_3...q_n$ or $0\bar{q}_2\bar{q}_3...\bar{q}_n$, based on which one has the smaller distance to q.

Corollary 2. *Suppose $p, q \in AQ_n$.*

1. *For $n = 1, 2$, $d(p, q) = 1$*
2. *For $n \geq 3$,*

$$d(p,q) = \begin{cases} d(p_2p_3...p_n, q_2q_3...q_n) & \text{if } p_1 = q_1 \\ 1 + min\{d(p_2p_3...p_n, q_2q_3...q_n), d(p_2p_3...p_n, \bar{q}_2\bar{q}_3...\bar{q}_n)\} & \text{if } p_1 \neq q_1 \end{cases}$$

Theorem 6. *The diameter of AQ_n is $\lceil n/2 \rceil$ for all $n \geq 1$ [10].*

Proposition 5. *There are $2n - 1$ node disjoint path between any two nodes p and $q \in AQ_n$, for all $n \geq 4$ [10].*

3.2.3 Routing Algorithm through Shortest Path

For the routing algorithm in [10], weights are assigned to the edges of AQ_n as follows:

1. $2i - 1$, if (p, q) is a hypercube edge with $p_i \neq q_i$, $1 \leq i \leq n$
2. $2j$, if (p, q) is a complement edge with $p_i = q_i$ for every i,$1 \leq i \leq j - 1$ and $p_i = \bar{q}_i$ for every i, $j \leq i \leq n$

Fig.3.1 shows an example for assigning weights to the edges in a path.

In [10] Theorem 5 is used to send a message through a shortest path from a source node, $s = s_1s_2...s_n$, to a destination, $t = t_1t_2...t_n$ in AQ_n. Three tasks, computing a tag, scanning a tag and routing to the next vertex, are done by any "current"

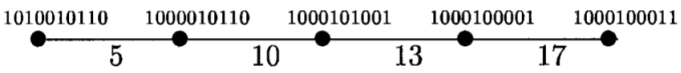

Figure 3.1: Associate weights with the edges in AQ_{10}

vertex, $c = c_1c_2...c_n$ while moving along a path. At each vertex c, tag is equal to $(c_1 \oplus t_1, c_2 \oplus t_2, ..., c_n \oplus t_n)$.

Algorithm 3 Routing Algorithm for AQ_n

Finding a shortest path from a source node, s, to a destination node, t;

$tag \longleftarrow s \oplus t$

1. **If** $tag = 00...0$
   - STOP; the message has reached t
2. **Else do:**
   - Scan the tag from left to right for the least entry, $tag(i)$ which is 1
   - **If** $tag(i+1) = 0$ **do**:
     - $tag(i) \longleftarrow 0$,
     - Transmit the message along the edge of weight $2i - 1$
   - **Else do:**
     - $tag(i) \longleftarrow 0$,
     - $tag(j) \longleftarrow \overline{tag(j)}$, $i < j \leq n$,
     - Transmit the message along the edge of weight $2i$

Example. Following the above procedure, a routing path from 000000 to 101011 in AQ_6 will be as follows:

$000000 \xrightarrow{1} 100000 \xrightarrow{5} 101000 \xrightarrow{10} 101011$.

The numbers above the arrows are weights of edges used for traveling from source node to the destination node.

Comparing the algorithms of AQ_n and Q_n, the only difference is that for any current vertex of AQ_n, one more step is required to check $tag(i+1)$ [10].

Chapter 4

One to Many Disjoint Paths Arrangement Graph

4.1 Introduction

In this chapter first we show that by using the same approach used for the star graph in [26], that is studied in the first chapter of this thesis, we can have n disjoint paths between any two nodes, when $1 < k < n - 1$.

Then we prove that there exist $k(n-k)$ disjoint paths in the (n,k)-arrangement graph from a source node to $k(n-k)$ different target nodes. The number of these node disjoint paths is equal to the degree of the graph. As mentioned before, due to symmetry in arrangement graphs, routing between two arbitrary nodes can be mapped to routing from the identity node to an arbitrary node.

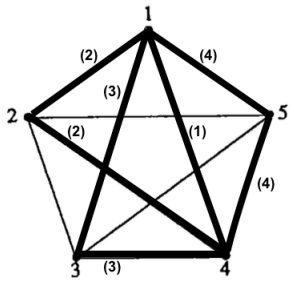

Figure 4.1: 4 disjoint paths in an $A_{5,1}$

4.2 Finding $n-1$ or n Disjoint Paths Between Two Nodes

The arrangement graph is a family of undirected graphs that contains the star graph family. For $k = n - 1$, the $(n, n - 1)$-arrangement graph is isomorphic to the n-star and for $k = 1$, the $(n, 1)$-arrangement graph is isomorphic to the complete graph with n nodes.

For $1 \leq k \leq n - 1$ we consider following cases in order to find disjoint paths.

Case 1: $k = 1$

Here the graph is a complete graph with n nodes with degree $n - 1$. So the maximum number of disjoint paths between two nodes would be $n - 1$. In this case all nodes are directly connected to each other and distance between any two nodes is equal to 1. So one path is a direct link between the source and the destination. Other $n - 2$ disjoint paths can be found by first going to other $n - 2$ neighbors of the source node and then routing to identity node through the direct link connecting them to each other. Fig.4.1 shows four disjoint paths from node 4 to node 1 in an $A_{5,1}$. These four paths are specified by numbers $1, 2, 3$, and 4.

Thus for Case 1, we have $n - 1$ disjoint paths where one path has length 1, while $n - 2$ paths have length 2.

Case 2: $k = n - 1$

Here the graph is an n-star graph with $n!$ nodes with degree $n-1$. So the maximum number of disjoint paths between two nodes would be $n-1$. In this case routing is exactly the same as the star graph, so Algorithm 2 which is discussed in Chapter 1, is used to find all $n-1$ disjoint paths.

Thus for Case 2, we have $n-1$ disjoint paths such that no path has length more than $d(s,t)+4$.

Case 3: $1 < k < n-1$

Here the (n,k)-arrangement graph has degree $k(n-k) \geq n$ for $n \geq 4$. We route to different sub-graphs for finding disjoint paths. By fixing each element of $[n]$ in the last position, we can have n sub-graphs, $A_{n-1,k-1}$. We use $*i$ to show a permutation whose last symbol is i, and $*$ represents any permutation of the $k-1$ symbols in $\{1,2,...,n\} - \{i\}$. Therefore, we have n sub-graphs with nodes of the forms $*1, *2, *3, ..., *n$ respectively.

If $d(t, I_k) = 1$ there exists a j that is not in its correct position in t, i.e $t_j \neq j$ and is replaced by an element from $\{k+1, k+2, ..., n\}$. One path is $t \rightarrow I_k$ with length one by placing j in its correct position. Other $k(n-k)-1$ paths are built by using the disjoint cycles with length no more than 6.

$k-1$ cycles of length 6, having the same external element as the source node, $(k+a)$ where $1 \leq a \leq n-k$, are as follows. We name them as cycles of the form (i):

$123...(i-1)i(i+1)...(j-1)(k+a)(j+1)...k \rightarrow$

$123...(i-1)j(i+1)...(j-1)(k+a)(j+1)...k \rightarrow$

$123...(i-1)j(i+1)...(j-1)i(j+1)...k \rightarrow$

$123...(i-1)(k+a)(i+1)...(j-1)i(j+1)...k \rightarrow$

$123...(i-1)(k+a)(i+1)...(j-1)j(j+1)...k \rightarrow$

$123...k$

In [22] it is proved that for $1 \leq i < j \leq k$ and any node p, any of its i-th neighbors and any of its j-th neighbors form a cycle of length six. It is also proved that any two

cycles with distinct $1 \leq i_1, j_1, i_2, j_2 \leq k$ are disjoint except at p. In our case all $k-1$ cycles are all disjoint except at p and I_k, since at first step of each routing a different i is replaced by j, for all $1 \leq i \leq k$, $i \neq j$. In the next step the foreign element is replaced by this i, that cause any two 6-cycles be disjoint. Then the foreign element, $k+a$, is fixed in t_i till routing to I_k. Therefore, all the cycles are disjoint.

$n-k-1$ cycles of length 3 are obtained by replacing $(k+a)$ with other foreign elements $k+b$, where $1 \leq a, b \leq n-k, b \neq a$. We name them as cycles of the form (ii):

$$123...(i-1)i(i+1)...(j-1)(k+a)(j+1)...k \rightarrow$$
$$123...(i-1)i(i+1)...(j-1)(k+b)(j+1)...k \rightarrow$$
$$123...(i-1)i(i+1)...(j-1)j(j+1)...k \rightarrow$$

for $1 \leq i \leq k, i \neq j$.

These $n-k-1$ cycles are all disjoint since in each of them $(k+a)$ is replaced by a different foreign element.

The rest of the cycles have length 4 and they are obtained by placing each of the other foreign elements in all t_i's, for $1 \leq i \leq k, i \neq j$. We name them as cycles of the form (iii):

$$123...(i-1)i(i+1)...(j-1)(k+a)(j+1)...k \rightarrow$$
$$123...(i-1)(k+b)(i+1)...(j-1)(k+a)(j+1)...k \rightarrow$$
$$123...(i-1)(k+b)(i+1)...(j-1)j(j+1)...k \rightarrow$$
$$123...(i-1)i(i+1)...(j-1)j(j+1)...k \rightarrow$$

These $k(n-k)-(k-1+n-k-1)-1$ cycles are also disjoint because each foreign element is fixed in one specific position, t_i, in each of the cycles. The cycles are obviously disjoint when different foreign elements are fixed in the same position t_i during routing.

For $A_{4,2}$, 4 disjoint paths from node 14 to identity node, 12, are shown in Fig.4.2.

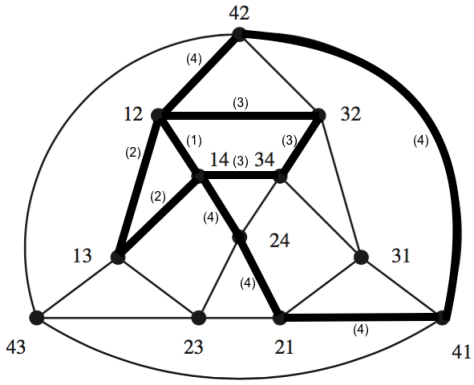

Figure 4.2: 4 disjoint paths in an $A_{4,2}$

Path number 1 is a shortest path with length 1, we just put 2 in its correct position t_2 . Path number 2 has length 2 which is of the form ii. Path number 3 has length 3 which is of the form iii. Path number 4 is the longest path, using a 6-cycle for routing of the form i, so its length is 5.

If $d(t, I_k) > 1$, for finding n node disjoint paths we will route to $n-1$ sub-graphs, other than the one where t is located in. These disjoint paths all go to identity node from its n different neighbors. Let $t = t_1t_2...t_k$. In no more than two steps, by changing the k-th element of t, it is routed to $n-1$ sub-graphs having elements $\{1, 2, ..., n\} - \{t_k\}$ in their last position. Now in each of these n sub-graphs routing will resume as follows:

Case 1: position $k \neq k$

The node is located in an $A_{n-1,k-1} = x_k$ with element $x \in \{1, 2, ..., n\} - \{k\}$ for its last element of its nodes. The path is determined by correcting its cycles, which is done by an $O(n)$ liner time greedy algorithm, to find the shortest route to either of the following nodes:

- $123...x$, a neighbor of I_k, when $k < x \leq n$
- $123...(j-1)(k+a)(j+1)...x$, a node that is two steps away from I_k, when $x = j, 1 \leq j \leq k$ and $1 \leq a \leq n-k$

These routings are all within $A_{n-1,k-1} = x_k$.

Case 2: position $k = k$

The node has k in its last position which means it is located in the same sub-graph as identity node. A path is determined by routing to p, a neighbor of I_k whose $p_j = (k+b) \neq j$ for a $1 \leq j < k$ and $1 \leq b \leq n-k, b \neq a$. Again following the greedy algorithm takes $O(n)$ time and all nodes of this path is within $A_{n-1,k-1} = k_k$.

This procedure is illustrated in Fig.4.3 for an $A_{n,k}$ and $t = *k$ which means it is located in the same sub-graph as identity node. Exterior dotted lines show routings to $n-1$ sub-graphs with elements $\{1, 2, ..., n\} - \{k\}$ in their last position. Dotted lines inside the sub-graphs, show routings to n nodes with distance 2 or 1 to identity node. Solid lines between sub-graphs show one step, which is replacing k in the last position of the nodes. $n - k$ of these nodes are routed to identity node. The remained ones are routed to k different nodes in the same sub-graph as identity node with distance 1 to identity node. These nodes are also routed to the I_k in one step by correcting its i-th position $i, 1 \leq i < k$.

If t is located in a sub-graph other than the one with k as its nodes last element, the procedure would be the same, except the dotted lines between sub-graphs will be from the sub-graph having t to other $n-1$ sub-graphs. The routings inside each sub-graph and the steps after that are all same as this case.

Routing in sub-graphs would result paths with a same length as $d(t, I_k)$. Let $p = p_1p_2...p_k$ and $q = (k+a)23...1$. Cyclic representation of q is $C_1 = (k, 1, (k+a))$ then $qq^{-1} = I_k$ where q^{-1} is shown with a cycle $C_1^{-1} = ((k+a), 1, k)$. In general $(k+a)$ is placed in $i-$th position and i is placed in the last position, $12..(i-1)(k+a)(i+1)..i$. Since $d(p, q) = d(pq^{-1}, I_k)$, pq^{-1} will fix two elements in positions i and k. Now comparing p and pq^{-1}, $k-2$ positions remain to be fixed. Suppose in the cyclic representation of p there were t cycles with e external ones, containing m elements in total. For pq^{-1}, i will be considered as k and $(k+a)$ will be considered as i.

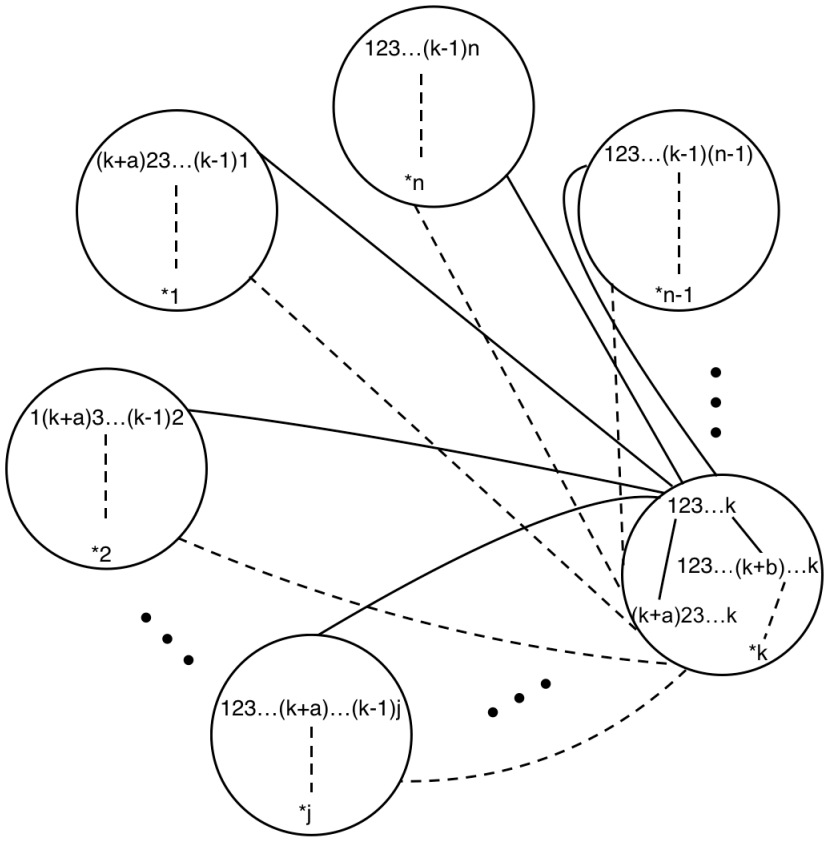

Figure 4.3: Finding n disjoint paths
in the $A_{n,k}$ for $1 < k < n$

If the $(k+a)$ is used in permutation of p but not in i-th position, in the cyclic representation the number of external cycles and the number of elements will be reduced by one. So we have $d(pq^{-1}, I_k) = (t-1)+(m-1)-2(e-1) = t+m-2e = d(p, I_k)$.

If it is already in i-th place, the total number of elements will be reduced by two as well as deduction in number of external cycles and total number of cycles, $d(pq^{-1}, I_k) = (t-1)+(m-2)-2(e-1) = t+m-2e-1 = d(p, I_k)-1$.

If the $(k+a)$ is not used in permutation of p, one external cycle will be added and the total number of elements will be one more as well, so we have, $d(pq^{-1}, I_k) = (t+1)+(m+1)-2(e+1) = t+m-2e = d(p, I_k)$.

The fact that the greedy algorithm is applied in different $A_{n-1,k-1}$ guarantees that these paths are disjoint.

Now by replacing the proper element in the last position, and position j for the node in the same sub-graph as identity node, we will have all n disjoint paths with lengths no more than $d(t, I_k)+2$.

Consequently, we have n disjoint paths where no path has length more than $d(p, I_k)+4$.

In all cases, the time complexity is $O(n^2)$ since $O(n)$ time is needed to route to different nodes in n different sub-graphs. Then the $O(n)$ linear-time greedy algorithm is applied to route to the identity node.

4.3 Finding Disjoint Paths from a Source to $k(n-k)$ Targets

As discussed in previous chapters, the (n,k)-arrangement graph, $1 < k < n-1$, is a regular graph with degree $k(n-k)$. Here we present a proof that there exist disjoint paths from a single source node to $k(n-k)$ targets. This theorem always holds for

$(k-1)! > n-k$. If $(k-1)! \leq n-k$, targets cannot have only non-foreign elements since $k!$ is less than $k(n-k)$. In this case routings have to include Case 2 and/or Case 4.1 besides other cases.

Theorem 7. *When T is a set of $k(n-k)$ distinct nodes in $A_{n,k}$, where $1 < k < n-1$, and $I_k \notin T$, there are $k(n-k)$ node disjoint paths from I_k to each of the nodes in T. Furthermore, all paths have length no more than diameter $+ (n-k) + 2$, diameter is considered as the maximum distance between any two nodes in the graph.*

Proof. First we route to all $k(n-k)$ neighbors of the source node, replacing each of the k symbols of I_k with one of the $n-k$ foreign symbols, $k+1, k+2, ..., n$. We mark them as the new sources, $s_1, s_2, ..., s_{k(n-k)}$.

Now we pair each source with a target node, considering the minimum distance and comparing position of the foreign element in new sources and targets.. This can be done by having minimum distances from each source to each target in a $k(n-k)$ by $k(n-k)$ matrix. First the lowest number in this matrix is specified, we name it *min*. We select one *min* and pair its corresponding source and target according to the following cases:

- If there is just one *min* in whole matrix, we select it.
- If more than one line have *min*:
  - We select the one in a line containing just one *min*.
  - If each line has *min* in two or more positions, the line containing less than the other lines is selected.
    - We select the one from this line such that its column contains no other *min* or less than the other columns.
- If we have more than one option with equal situations, the foreign element and its position in the source is compared with the targets. The target that has a

same foreign element in a same position as this source is selected.

- If all options are the same, one of them is chosen randomly.

We use an example to illustrate each step. Considering $A_{9,7}$, we can have $7(9-7) = 14$ disjoint paths from I_k to 14 different target nodes given in bellow matrix. Procedure starts by routing to 14 neighbors of I_k, considering these as new sources. Minimum distance from each source to each target is found. Results are given bellow.

$MinimumDistance =$

	1234568	1234569	1234587	1234597	1234867	1234967	1238567	1239567	1284567	1294567	1834567	1934567	8234567	9234567
6351742	8	8	8	8	8	8	8	8	8	8	8	8	8	8
4351792	8	8	8	8	8	8	8	8	8	8	8	8	8	8
6351798	6	7	7	6	8	7	7	8	8	7	8	7	7	8
6451792	7	8	7	6	7	8	7	8	7	8	7	8	7	8
6341792	7	8	7	6	7	8	7	8	7	8	7	8	7	8
6351782	7	7	7	8	7	7	9	8	7	7	7	7	9	8
8351792	7	7	8	7	7	7	9	8	7	7	7	7	7	9
6351794	7	8	7	6	7	8	7	8	7	8	7	8	7	8
6358792	7	7	8	7	7	7	7	8	7	7	7	7	8	9
6381792	7	7	7	6	8	7	7	8	6	7	8	7	7	8
6851792	8	7	7	6	8	7	7	8	9	7	6	7	7	8
6354792	6	6	8	6	6	6	8	8	6	6	6	6	7	8
6351892	8	7	7	6	6	7	7	8	8	7	8	7	7	8
6351492	7	8	7	6	7	8	7	8	7	8	7	8	7	8

For paring a source and a target, the minimum number in the whole matrix is specified. Here 6 is the lowest number in this matrix. There are nine sources having distance 6 with some targets. Two of these sources have only one 6 in their line, these two sources are 6351794 and 6351492. Both of these lines have distance 6 to a same target node 1234597, so one of them is chosen randomly. This procedure is done for paring all sources and target nodes. Results are as follows:

$MinimumDistance(1234597, 6351794) = 6$

$MinimumDistance(1234568, 6351798) = 6$

$MinimumDistance(1284567, 6381792) = 6$

$MinimumDistance(1834567, 6851792) = 6$

$MinimumDistance(1234867, 6351892) = 6$

$MinimumDistance(1934567, 6354792) = 6$

$MinimumDistance(8234567, 8351792) = 7$

$MinimumDistance(1238567, 6358792) = 7$

$MinimumDistance(1234587, 6351782) = 7$

$MinimumDistance(1294567, 6451792) = 8$

$MinimumDistance(1234569, 6351492) = 8$

$MinimumDistance(1234967, 6341792) = 8$

$MinimumDistance(9234567, 4351792) = 8$

$MinimumDistance(1239567, 6351742) = 8$

For each s_i its foreign element is kept in the same position during routing. We have following cases:

Case 1. If the target has just non-foreign element, we will have upto $k(n-k)$ node disjoint paths in our routing according to $(k)!-1$ number of different nodes with permutations of non-foreign elements. These paths are disjoint since each path has one foreign element in one of the k positions, it is internal for this path, which makes it different from all other paths. Here all cycles are corrected and at last step the non-foreign element is placed in the string, which its position is taken by the foreign element during routing.

The length of path is equal to minimum distance selected from the matrix, $d(s_i, t_j)$ for each i, j that are paired. In this case there is one external cycle and l internal ones, $0 \leq l$. During the correction of the cycles, the foreign element, x, remains in the same position and routing is continued with the next cycle. If there is no internal

cycle routing is terminated with making the foreign element an external one. Let a_i be the non-foreign element which is external for s_i. We start correcting an external cycle $C_1 = (a_i, a_{12}, a_{13}, ..., a_{1m_1}, x)$, but x is not corrected. The procedure continues with correcting internal cycles and a_{1m_1} is considered as an external element. When cycles are all corrected we have a_{1m_1} as an external element. All along the path x is in a fixed position which makes this path distinct from all other paths. Now, after correcting all cycles, x is replaced by a_{1m_1}. So no extra step is done during routing and the length of the path is equal to $d(s_i, t_j)$.

In our example routing for $(1239567, 6351742)$ is done by Case 1 since the target node contains only non-foreign elements. The foreign element is fixed in its position during routing and is corrected at last step. 4 is the non-foreign element which is external for this source node. We start correcting the external cycle $C_1 = (4, 6, 1, 9)$ except 9. After correcting this cycle, 1 is external and we use it for correcting the internal cycle $C_2 = (2, 7, 5, 3)$. After correcting the internal cycle 1 is again an external element. Only one step is remained which was skipped before. We should replace 9 by 1.

$1239567 \rightarrow 1239547 \rightarrow 6239547 \rightarrow 6139547 \rightarrow 6139542 \rightarrow 6139742 \rightarrow 6159742 \rightarrow 6359742 \rightarrow 6351742$.

The length of this path is 8 which is equal to the shortest path between these two nodes. One step is done at the beginning for routing to this new source, $1234567 \rightarrow 1239567$. So the length of path from identity node to this target is $8 + 1 = 9$.

Case 2. There is only one foreign element which is in the same position as in its paired source. Routing contains external and/or internal cycles. Again let a_i be the non-foreign element which is external for s_i. If there is no external cycle a_i is considered as an external element for correcting internal cycles. If there is an external cycle only a_i can be its first element and its last element is considered as an external element for internal cycles. The length of the path here again is equal to $d(s_i, t_j)$.

Two routings $(1234587, 6351782)$ and $(1234597, 6351794)$ of our example are done by this case. The position of the only foreign element is the same in each of these pairs. Both of the routings contain external and internal cycles, and 6 is the external element for both of them. The first one has the external cycle $C_1 = (6, 1, 4)$ and the internal cycle $C_2 = (2, 7, 5, 3)$. The routing begins with correction of the external cycle and its last element is considered as an external element to correct the internal cycle as follows:

$1234587 \rightarrow 6234587 \rightarrow 6231587 \rightarrow 6431587 \rightarrow 6431582 \rightarrow 6431782 \rightarrow 6451782 \rightarrow 6351782$.

The second routing with one external cycle $C_1 = (6, 1, 4, 7, 5, 3, 2)$ will be done following the same procedure:

$1234597 \rightarrow 6234597 \rightarrow 6231597 \rightarrow 6231594 \rightarrow 6231794 \rightarrow 6251794 \rightarrow 6351794$.

The length of the path for routing from the identity node to first one is $7+1 = 8$. The length of the second path is also calculated in a same way which is equal to $6+1 = 7$.

Case 3. There is only one foreign element in a target which is not in the same position as in its paired source or the foreign element in s_i is not the one in target, the same procedure as Case 1 is done and at the end in one step the foreign element is placed in the right position. The only difference with the routing in Case 1 is that If the foreign element was in the middle of an external cycle, when it is met, the routing is continued with correcting the next cycle and after correcting all cycles, correction of the cycle containing the foreign element is proceed with the next element of the foreign element till the last element of the cycle. At last with no more than two steps, we will reach to a node with all non-foreign elements and the non-foreign external element in the position of the foreign element. For internal cycles we can shift the elements till the foreign element becomes the last element in this cycle. Then the foreign element is put in its correct position. These are again node disjoint paths, since Case 1 shows that there exist disjoint paths from identity node to at most $k(n-k)$ different nodes

which only have non-foreign elements in their representations. If the foreign element is the last element of the cyclic representation of the target node, no more step is needed in addition to corrections of internal and/or external cycles. Hence the length of the path is $d(s_i, t_j)$. If the foreign element is not in the last two elements of the external cycle, the length of the path is $d(s_i, t_j) + 2$.

Following Case 3 we can find the disjoint paths for each of following pairs in our example.

$(9234567, 4351792)$

$(1294567, 6451792)$

$(1234569, 6351492)$

$(1234967, 6341792)$

$(1934567, 6354792)$

Each source has only one foreign element that is not in the same position as the target it is paired with. By the same procedure as Case 1, each source node is routed to a node with all non-foreign elements and the only foreign element, which is used during the routing, is placed in its correct position at last step.

For the first pair, the external cycle is $C_1 = (1, 4, 9, 6)$ and the internal cycle is $C_2 = (5, 3, 2, 7)$.

$9234567 \rightarrow 9231567 \rightarrow 9231467 \rightarrow 9251467 \rightarrow 9351467 \rightarrow 9351462 \rightarrow 9351762 \rightarrow 4351762 \rightarrow 4351792$.

Routing for next 3 pairs are as follows:

$1294567 \rightarrow 1294537 \rightarrow 6294537 \rightarrow 6291537 \rightarrow 6491537 \rightarrow 6491532 \rightarrow 6491732 \rightarrow 6451732 \rightarrow 6451792$

$1234569 \rightarrow 1234579 \rightarrow 6234579 \rightarrow 6231579 \rightarrow 6231479 \rightarrow 6251479 \rightarrow 6351479 \rightarrow 6351472 \rightarrow 6351492$

$1234967 \rightarrow 1234957 \rightarrow 6234957 \rightarrow 6231957 \rightarrow 6241957 \rightarrow 6341957 \rightarrow 6341952 \rightarrow 6341752 \rightarrow 6341792$

The cyclic representation for the last pair in this case contains only one external cycle $C_1 = (2, 7, 5, 3, 9, 6, 1)$. Here 9, the foreign element, is in the middle of the cycle. So, we consider the cycle without 9. By making 1 an external element, we use it to correct 3 and 9. First 3 is replaced by 1, then 3 is placed in its correct position. At last 9 is placed in its correct position and 1 is made an external again.

$1934567 \rightarrow 1934562 \rightarrow 1934762 \rightarrow 1954762 \rightarrow 1954732 \rightarrow 6954732 \rightarrow 6954712 \rightarrow 6354712 \rightarrow 6354792$

Correction of this cycle needs 2 more steps which will cause a path with length $d(1934567, 6354792) + 2 = 6 + 2 = 8$. So the length of the path from the identity node to this target is $8 + 1 = 9$. The length of other paths is equal to $d(s_i, t_i) + 1$ for each s_i and t_i that are paired.

Case 4. If there are more than one foreign element in a target, the foreign element in its paired new source is fixed from beginning of routing as in Case 1. Consider z as the number of foreign elements and x as a foreign element used in routing. Whether this foreign element is in a correct position or not, we have one of the following cases:

Case 4.1. If the foreign element x is in its correct position, i, it is routed to a node with $k - 1$ non-foreigns, including internals for the target and non-foreign external ones, and also a foreign element in its correct positions. Then the remaining foreign elements are replaced one by one in a sequence that resulting node is not met before.

The length of path is at most $d(s_i, t_j) + (z - 1)$. During routing no foreign element, other than the first one, is used. If i, which is replaced by x, is an internal element for the target we have z external cycles, if not each foreign external element forms an external cycle. So the routing starts with external cycle with a_i as its first element. Then correction of internal cycles begins. After correcting internal cycles, correction of external cycles begins from their second element, since we do not want to replace any foreign element yet. The last element of each external cycle (which is a non-

foreign one) is considered as an external element for correcting next external cycle from its second element. At last, all external elements are placed in their correct positions, so $z-1$ extra steps are done while correcting external cycles.

In our example the routings for remained targets are calculated by Case 4.1.

$(1234568, 6351798)$

$(8234567, 8351792)$

$(1238567, 6358792)$

$(1284567, 6381792)$

$(1834567, 6851792)$

$(1234867, 6351892)$

In each of these sources there is a foreign element which is in its correct position comparing to its paired target node. So during routing this foreign element is fixed in its position and at last step the other foreign element is replaced. Routing for the first pair in this case, $(1234568, 6351798)$, is done by two external cycles $C_1 = (9, 6, 1, 4)$, $C_2 = (7, 5, 3, 2)$. One external cycle is formed by one foreign element and the other one is formed by 7, which is a non-foreign element but it is replaced by 8 in 1234568. So routing will start by correcting C_2 then C_1 from its second element. At last the 9 is put in its correct position.

$1234568 \rightarrow 1234768 \rightarrow 1254768 \rightarrow 1354768 \rightarrow 1354728 \rightarrow 6354728 \rightarrow 6351728 \rightarrow 6351798$

Rest of the paths are as follows:

$8234567 \rightarrow 8231567 \rightarrow 8431567 \rightarrow 8431562 \rightarrow 8431762 \rightarrow 8451762 \rightarrow 8351762 \rightarrow 8351792$

$1238567 \rightarrow 1238547 \rightarrow 6238547 \rightarrow 6138547 \rightarrow 6138542 \rightarrow 6138742 \rightarrow 6158742 \rightarrow 6358742 \rightarrow 6358792$

$1284567 \rightarrow 1384567 \rightarrow 1384562 \rightarrow 1384762 \rightarrow 1384752 \rightarrow 6384752 \rightarrow 6381752 \rightarrow 6381792$

1834567 → 1834562 → 1834762 → 1854762 → 1854732 → 6854732 → 6851732 → 6851792

1234867 → 1254867 → 1354867 → 1354862 → 1354872 → 6354872 → 6351872 → 6351892.

In this case length of the paths is one more than the minimum distance for each pair, since there is only one other foreign element that should be replaced. The length of the longest path in this part is $7 + 1 = 8$. So with 9 steps identity node is routed to this target node.

Case 4.2. If the foreign element x is not in its correct position or target node does not have x, first we route to a node with all non-foreign elements, as in Case 1 then start replacing foreign elements in a sequence that resulted node is not met before.

The length of path here is at most $d(s_i, t_j) + z$. As in the Case 1, no other foreign element is used during routing and if $i = a_i$ is an internal element we have z external cycles, if not each foreign element forms an external cycle. So the routing starts with the external cycle with a_i as its first element. After correcting internal cycles, correction of external cycles begins from their second element, since we do not want to replace any foreign element yet. Last element of each external cycle (which is a non-foreign one) is considered as an external element for correcting next external cycle from its second element. At last, x is replaced by an external non-foreign one. During routing x remains in a fixed position and procedure continues with its next element. Same as Case 3 by no more than two steps we reach to a node with all non-foreign elements. All external elements are placed in their correct positions, so $(z - 1) + 2 = z + 1$ extra steps are done during the procedure to reach the target. If x is not a target's element, it must be a last element of an external cycle which is replaced by a non-foreign element when all cycles are corrected, except replacing the foreign elements. Then z foreign elements through z steps are replaced to reach the

target.

Since sequences of all non-foreign elements are unique in each of these routings and only one foreign element has been used for each, all paths are node disjoint.
The maximum value of distance between two arbitrary nodes is equal to diameter of the graph, so we will consider diameter $= \lfloor 3/2k \rfloor$ as the maximum for $d(s_i, t_j)$. Also one step should be added to all distances since we start routing from fist neighbors of identity node. So length of longest path is $\lfloor 3/2k \rfloor + z + 1 + 1$ where $z \leq n - k$.

In our example the length of the longest path in all cases is 9. This is less than the worst case which is $8 + 2 + 2 = 12$. □

4.3.1 Example

Example. Suppose that in $A_{5,3}$, we want to find $3(5-3) = 6$ disjoint paths from the identity node to six targets.

$t_1 = 314, \quad t_2 = 514, \quad t_3 = 234, \quad t_4 = 254, \quad t_5 = 213, \quad t_6 = 215.$

First we route to all 6 neighbors of the identity node:

$s_1 = 124, \quad s_2 = 125, \quad s_3 = 143, \quad s_4 = 153, \quad s_5 = 423, \quad s_6 = 523.$

Now we should calculate distance between any source and target to pair them based on the minimum distance and position of its foreign element.

$MinimumDistance =$

	314	514	234	254	213	215
124	2	2	2	2	4	4
125	3	3	3	3	4	3
143	4	3	4	3	2	3
153	3	4	3	2	2	3
423	4	3	4	3	2	3
523	3	2	3	4	2	3

Now we start selecting the lowest distance from matrix, when a source and a target are paired, related row and column will be omitted. Bellow are 6 pairs based on the procedure:

$MinimumDistance(143, 213) = 2$

$MinimumDistance(153, 254) = 2$

$MinimumDistance(523, 514) = 2$

$MinimumDistance(124, 314) = 2$

$MinimumDistance(125, 215) = 3$

$MinimumDistance(423, 234) = 4$

The routing procedure for each pair is as follows:

$143 \to 243 \to 213$, Case 2

$153 \to 253 \to 254$, Case 4.1

$523 \to 513 \to 514$, Case 4.1

$124 \to 324 \to 314$, Case 2

$125 \to 135 \to 235 \to 215$, Case 2

$423 \to 421 \to 431 \to 231 \to 234$, Case 3

By adding 1 to each path, the length of paths from the source node to target nodes will be calculated. Here paths has no collision and each node has been used once. Length of the longest path is $4 + 1 = 5 \leq 4 + 1 + 2 = 7$ and Theorem 7 holds.

Example. In a $(4, 2)$-arrangement graph we want to find disjoint paths from source to 4 target nodes

$t_1 = 21, \quad t_2 = 31, \quad t_3 = 42, \quad t_4 = 43.$

We go over the procedure. First we find all the neighbors of the identity node.

$s_1 = 13, \quad s_2 = 14, \quad s_3 = 32, \quad s_4 = 42.$

And then we build a 4×4 matrix for pairing them:

$$MinimumDistance = \begin{array}{c|cccc} & 21 & 31 & 42 & 43 \\ 13 & 2 & 3 & 2 & 1 \\ 14 & 2 & 2 & 2 & 2 \\ 32 & 2 & 1 & 1 & 2 \\ 42 & 2 & 2 & 0 & 1 \end{array}$$

These are paired as bellow:

$MinimumDistance(42, 42) = 0$

$MinimumDistance(32, 31) = 1$

$MinimumDistance(14, 21) = 2$

$MinimumDistance(13, 43) = 1$

Paths are as follows:

42 (this node is a neighbor of source and a target)

$32 \rightarrow 31$, Case 2

$14 \rightarrow 24 \rightarrow 21$, Case 1

$13 \rightarrow 43$, Case 4.1

And the length of the longest path here is equal to $2 + 1 = 3 \leq 3 + 2 + 2 = 7$.

Chapter 5

Proposed Algorithm

Augmented Cubes

5.1 Introduction

In this chapter, we present an algorithm for finding disjoint shortest paths from a source node to $2n-1$ targets in an augmented cube. The number of these paths is maximum since it is equal to the degree of the graph. As mentioned in Chapter 3 we can safely map the source node to the identity node, and then specify all target nodes based on this mapping. Thus routings will be from a special node 0^n to $2n-1$ other nodes. More formally, it can be stated as follows:

Given a source node and $2n-1$ target nodes in an $n-$dimensional augmented cube, do there exist $2n-1$ shortest and disjoint paths from source to all of these targets?

First we explain our approach in detail then present and discuss a polynomial algorithm.

5.2 Finding Disjoint Shortest Paths in the Hypercube

The problem of finding the maximum number of disjoint and shortest paths have been studied on the hypercube in [8]. The hypercube is a similar structure to the augmented cube. So we will review the result for the hypercube first.

Definition 15. *A permutation of the elements of a finite set is called an ordering. Suppose P and Q are two sets ordered as $O_P = (p_1, p_2, p_3, ..., p_k)$ and $O_Q = (q_1, q_2, q_3, ..., q_l)$ where $k = |P|$ and $l = |Q|$. We say that O_P and O_Q are disjoint if*

$$\{p_1, p_2, p_3, ..., p_i\} \neq \{q_1, q_2, q_3, ..., q_i\}$$

for $1 \leq i \leq min(k, l)$ unless $i = |P| = |Q|$.

For a permutation $P = [p_1, p_2, p_3, ..., p_n]$, we use the following notations which are defined in [14]:

- $[P_k] = [p_1, p_2, p_3, ..., p_k]$, the k elements of P in the same order,
- $\{P_k\} = [p_1, p_2, p_3, ..., p_k]$, the set of the first k elements in P.

Example. If $P = [3, 1, 4, 2]$ then $[P_3] = [3, 1, 4]$ and $\{P_3\} = \{1, 3, 4\}$.

Definition 16. *A node $v = v_1v_2v_3...v_n$, $v_i \in \{0, 1\}$ in an n-cube is presented by $X = \{i | v_i = 1, 1 \leq i \leq n\}$. In other words, X is the set of all dimensions (bits) in v that are 1.*

Example. In a 4-hypercube, the set representation of three nodes are given bellow:

$v_1 = 0011 \qquad X_1 = \{3, 4\}$

$v_2 = 0101 \qquad X_2 = \{2, 4\}$

$v_3 = 1110 \qquad X_3 = \{1, 2, 3\}$

Definition 17. *A path in an n-cube in the form of* $< 0^n, v_1, v_2, ..., v_{n-1} >$ *can be shown as a permutation* $P = [p_1, p_2, p_3, ..., p_n]$ *such that* $\{P_k\} = X_k$*, where* X_k *is the set representation of* v_k*.*

Example. In a 4-hypercube, a path from $s = 0000$ to $t = 1111$ can be represented as follows:

$< 0000, 0010, 0110, 1110, 1111 > \Longleftrightarrow P = [3, 2, 1, 4]$.

An ordering of a binary representation of a node is a shortest path from source node to this node since its length is equal to the number of 1s in the binary representation. At each step a dimension with a 1 is corrected by going to a node that has 1 in this dimension.

Definition 18. *Let* $(P_1, P_2, P_3, ..., P_m)$ *be a collection of subset of a set* $P = \{p_1, p_2, p_3, ..., p_n\}$, $m \leq n$*. An ordered set of distinct elements* $[p_{i_1}, p_{i_2}, ..., p_{i_m}]$ *is called a system of distinct representatives (SDR) if* $p_{i_j} \in P_j$*, for* $1 \leq j \leq m$*.*

For example, if $P = \{1, 2, 3, 4\}$ and $P_1 = \{1, 2\}$, $P_2 = \{3\}$, and $P_3 = \{2, 3\}$ then $[1, 3, 2]$ is an SDR for P_1, P_2, and P_3. But if we change P_2 and P_3 to $P_2 = \{1\}$ and $P_3 = \{2\}$ there does not exist an SDR for them. It is obvious that it is not always possible to find an SDR for some sets.

In [8] it is shown that an SDR of $(P_1, P_2, ..., P_m)$ corresponds to a matching of size m of the bipartite graph where the partite sets are P and $\{P_1, P_2, ..., P_m\}$ such that there is an edge from $p \in P$ to P_i iff $p \in P_i$. The well-known Halls Theorem [15] gives the iff condition for the existence of an SDR for a collection of finite sets.

Theorem 8. *A collection of nonempty finite sets* $P_1, P_2, ..., P_n$ *has an SDR iff for any* $1 \leq i_1 < i_2 < ... < i_k \leq n$, $|\cup_{j=1}^{k} P_{i_j}| \geq k$ *[15].*

For the previous example there is no SDR since $|P_1 \cup P_2 \cup P_3| = 2 \ngeq 3$.

For n target nodes in an n-cube with 2^n nodes:

$$t_1 = t_{11}t_{12}...t_{1n},$$
$$t_2 = t_{21}t_{22}...t_{2n},$$
$$.$$
$$.$$
$$.$$
$$t_n = t_{n1}t_{n2}...t_{nn}$$

they correspond to n sets $T_1, T_2, ..., T_n$ which are defined as follows. If $t_{ij} = 1$, then we include j in set T_i, which means we include the numbers corresponding to the bits of t_i that are 1 in set T_i. It is shown in [14] that a necessary and sufficient condition for the existence of n disjoint shortest paths in the n-cube is that there exists an SDR for binary representation of n target nodes.

Theorem 9. *For any collection of nonempty finite sets $T_1, T_2, ..., T_s$, in which all singletons are distinct, there is a disjoint ordering if and only if a SDR exists for the collection.*

For example, for nodes

$$t_1 = 0110$$
$$t_2 = 1000$$
$$t_3 = 0111$$
$$t_4 = 1001,$$

we have

$$T_1 = \{2, 3\}$$
$$T_2 = \{1\}$$
$$T_3 = \{2, 3, 4\}$$
$$T_4 = \{1, 4\}.$$

One SDR for the four sets is $[2, 1, 3, 4]$.

The binary representations of the n nodes can be viewed as the adjacency matrix of a bipartite graph and the condition that an SDR exists means that Halls condition [15] is satisfied, i.e., n disjoint shortest paths exist if and only if there exists a permutation $i_1 i_2 ... i_n$ of symbols from $\{1, 2, ..., n\}$ such that $t_{1,i_1} = t_{2,i_2} = ... = t_{n,i_n} = 1$, namely, there is a 1 on each row and each column in the adjacency matrix. For the above example, it represents a bipartite graph $G = (\{A, B, C, D, a, b, c, d\}, \{(A, b), (A, c), (B, a), (C, b), (C, c), (C, d), (D, a), (D, d)\})$. One possible solution is specified with bold font bellow:

$$\begin{matrix} 0 & 1 & \mathbf{1} & 0 \\ \mathbf{1} & 0 & 0 & 0 \\ 0 & \mathbf{1} & 1 & 1 \\ 1 & 0 & 0 & \mathbf{1} \end{matrix}$$

This particular solution is equivalent to a perfect matching of $\{(A, c), (B, a), (C, b), (D, d)\}$. Note that the solution and thus the perfect matching problem is not unique. Finding a maximum (perfect) matching for a bipartite graph with $2n$ vertices can be done in $O(n^{5/2})$ [16].

In [8] also an algorithm with $O(n^3)$ time complexity is presented to find all n disjoint paths from a source to n target nodes in Q_n. Briefly, the paths are found by performing a row reduction, which can be done in $O(n)$ time, on a matrix of binary representations of the n nodes with an SDR. The reduction makes sure that no two paths meet in a node during routing. The major steps of the algorithm are:

1. Check to see whether an SDR exists

2. If yes, find a disjoint ordering

The procedure for finding a disjoint ordering is given in [8].

Finding an SDR takes $O(n^{5/2})$ time and finding a disjoint ordering for a given SDR takes $O(n^3)$ time.

Example. Given five sets A_i, $i = 1, 2, 3, 4, 5$ whose matrix representation is as follows:

$$\begin{array}{lccccc} A_1 : & \mathbf{1} & 1 & 0 & 0 & 0 \\ A_2 : & 0 & 1 & \mathbf{1} & 0 & 0 \\ A_3 : & 1 & \mathbf{1} & 1 & 0 & 0 \\ A_4 : & 0 & 0 & 0 & \mathbf{1} & 1 \\ A_5 : & 0 & 0 & 1 & 1 & \mathbf{1} \end{array}$$

with the bold ones as the SDR. Since A_3 has the highest Hamming weight, we do the reduction on A_3. We first try the first 1 (from the left) and the resulting set is A_2. Then we try the third 1 and the resulting set is A_1. We then switch the two 1's in the SDR in A_1 and A_3 and remove the second 1 in A_3 to get:

$$\begin{array}{lccccc} A_1 : & 1 & \mathbf{1} & 0 & 0 & 0 \\ A_2 : & 0 & 1 & \mathbf{1} & 0 & 0 \\ A_3 : & \mathbf{1} & 0 & 1 & 0 & 0 \\ A_4 : & 0 & 0 & 0 & \mathbf{1} & 1 \\ A_5 : & 0 & 0 & 1 & 1 & \mathbf{1} \end{array}$$

We now have a disjoint ordering as follows:

$$O_{A_1} = ()$$
$$O_{A_2} = ()$$
$$O_{A_3} = (2)$$
$$O_{A_4} = ()$$
$$O_{A_5} = ()$$

The next set to reduce is A_5. When the third 1 is removed, the resulting row becomes

A_4. So we remove the fourth 1 to obtain:

$$\begin{array}{lccccc} A_1 : & 1 & \mathbf{1} & 0 & 0 & 0 \\ A_2 : & 0 & 1 & \mathbf{1} & 0 & 0 \\ A_3 : & \mathbf{1} & 0 & 1 & 0 & 0 \\ A_4 : & 0 & 0 & 0 & \mathbf{1} & 1 \\ A_5 : & 0 & 0 & 1 & 0 & \mathbf{1} \end{array}$$

The disjoint ordering now becomes:

$$O_{A_1} = ()$$
$$O_{A_2} = ()$$
$$O_{A_3} = (2)$$
$$O_{A_4} = ()$$
$$O_{A_5} = (4)$$

The next five reductions are straightforward and we will end up with:

$$\begin{array}{lccccc} A_1 : & 0 & \mathbf{1} & 0 & 0 & 0 \\ A_2 : & 0 & 0 & \mathbf{1} & 0 & 0 \\ A_3 : & \mathbf{1} & 0 & 0 & 0 & 0 \\ A_4 : & 0 & 0 & 0 & \mathbf{1} & 0 \\ A_5 : & 0 & 0 & 0 & 0 & \mathbf{1} \end{array}$$

The disjoint ordering generated is:

$$O_{A_1} = (2, 1)$$
$$O_{A_2} = (3, 2)$$
$$O_{A_3} = (1, 3, 2)$$
$$O_{A_4} = (4, 5)$$
$$O_{A_5} = (5, 3, 4)$$

5.3 Finding $2n-1$ Disjoint Shortest Paths in AQ_n

As we know, an augmented cube graph with dimension n, has 2^n number of nodes. Two nodes p and q are connected

- by a hypercube edge if their binary representations only differ at one place, $p_l = \bar{q}_l$ and $p_i = q_i \ \forall i \neq l$, or
- by a complement edge if their binary representations are equal just for the first $l-1$ elements, $p_i = q_i$, $1 \leq i \leq l-1$ and $p_i = \bar{q}_i$ $l \leq i \leq n$.

The first step is to compute distances from the source node to all target nodes by using just hypercube edges. The results of this step will be needed when the same number of edges is needed to route to this node from the identity node by using complement edges and hypercube edges simultaneously.

In the second step, the routing algorithm presented in Chapter 3, Algorithm 3, is employed to determine weights of edges. We use a counter for the number of edges in Algorithm 3, to represent the distance between this node to identity node. The weights are in the from of $2i$ and $2i-1$ for $1 \leq i \leq n$, so the range of weights would be $1 \leq weights \leq 2n-1$. The weight cannot be equal to $2n$ since there is no bit after the last position and if a transmission is needed here it will be done by an edge weighting $2n-1$.

In the third step, for each node that has equal results calculated in Step 1 and Step 2, the bit numbers of the 1s in the binary representation are saved.

In the fourth step a matrix is created. We are going to convert the set of weights for each target node to a binary string of length $2n-1$ as follows: since the highest weight is $2n-1$, a zero binary string with length of $2n-1$ is considered. The bits in the columns i, for all $i \in$ set of weights of edges resulted from Step 2, are changed to 1, starting from left to right. All other bits remain 0. This is done for all $2n-1$ nodes. These binary strings can be written as a matrix, so we will have a $(2n-1) \times (2n-1)$ matrix.

For example a node $p = 011101$ in AQ_6 has distance 2 to the identity node, $d(011101, 000000) = 2$. The set of weights, W, for routing to this node contains two edges with weights: $w_1 = 2 \times 2 = 4$ and $w_2 = (2 \times 5) - 1 = 9$. Then we convert $W = \{4, 9\}$ to a binary string with length $2n-1$ which has 1 in the columns 4 and 9: 00010000100.

Step 5 checks if an SDR exists for this matrix or not. If there is an SDR, the algorithm goes to step 7 and finds a disjoint ordering which results in disjoint shortest path from the identity node to all target nodes. If there is no SDR for this matrix the procedure continues to the next step.

Step 6.1 First we check even columns of this matrix. If all bits in an i-th column are 0, for an even i $(1 \leq i \leq 2n-1)$, there is no SDR for this set so there cannot be disjoint shortest paths from the source node to these target nodes and the algorithm stops here. The reason for this is explained in next step.

Step 6.2. If this matrix does not have an even column with all zero bits we need to convert the saved hypercube routings to $2n-1$ bit strings. We use the results of Step 3 which are the bit numbers of the 1's in the binary representations of the nodes. Since only one bit is changed in each step, hypercube routing is the same as augmented cube routing, but only using edges with odd weights computed by $2i-1$. The result is a $2n-1$ binary string with 1 in odd columns. Thus if all bits in an

even columns of all augmented cube routings are zero, there will be no SDR for this set of target nodes since hypercube routings also do not have a non-zero bit in any of their even columns. Again, for each node that has a hypercube routing, a zero binary string, 0^{2n-1} is considered. We change the bits in columns $2i-1$ into 1, for all i's resulted from step 3.

Step 6.3. Now to check all possible combinations of the augmented cube routings and the hypercube ones, the rows of matrix are changed as follows. For each node in i-th row of matrix, $1 \leq i \leq 2n-1$, which has both routings, we combine both routings to have a binary string of length $2n-1$ with 1s in the columns where augmented cube routing and hypercube routing are 1. This binary string is placed in i-th row of the matrix.

Step 6.4. After updating the matrix, again we check the existence of an SDR for this matrix.

Step 6.5. If there is no SDR, it means that there cannot be disjoint shortest paths to these targets and our algorithm stops.

Step 6.6. If an SDR is found, the matrix is updated again. For lines with both routings, the routing that contains the the 1 forming the SDR, is kept in this matrix and the other one is deleted. If the 1 in the SDR is for both routings, it does not matter which one is deleted and which one is remained in the matrix.

Finally this $(2n-1) \times (2n-1)$ matrix with an SDR will result in finding shortest and disjoint paths to target nodes The whole process is the same as the one for hypercube in [8].

5.3.1 Proposed Algorithm

In these section the algorithm described earlier is presented.

Algorithm 4 Routing Algorithm for $2n-1$ target nodes in an AQ_n

Finding $2n-1$ node disjoint paths from a source node, s, to $2n-1$ destination nodes, $t_1, t_2, t_3..., t_{2n-1}$;

1. Compute distance to source node for all target nodes in hypercube
2. Compute weights of edges in augmented cube for routing to targets and distances to source node by using Algorithm 3
3. **If** hypercube distance is equal to augmented cube distance for a node
   - save hypercube routing for this node (save i where bit number i is 1 in binary representation of this node)
4. Create binary strings for all $2n-1$ augmented cube routings
5. Find SDR
6. **If** there is no SDR
   - 6.1. **If** there is an even column with no 1, STOP
   - 6.2. **Else** Create binary strings for the hypercube routings saved in Step 3
   - 6.3. Update each row of the matrix
     - For nodes having the same length augmented cube routing and hypercube routing, combine them and update this row of matrix
   - 6.4. Find SDR
   - 6.5. **If** there is no SDR, STOP
   - 6.6. **Else**
     - For nodes with both routings, keep the one with the bit 1 used in the SDR, delete the other one, and update this row of matrix
7. Perform reduction (same as hypercube)

5.3.2 Performance

The performance of each step of the proposed algorithm is as follows:

- Step 1. There are $2n-1$ target nodes with a binary representation of length n. Going through all bits of one binary string for finding the bits which are 1 takes $O(n)$ time. So the entire checking for all nodes can be done in $O(n^2)$.

- Step 2. In this step Algorithm 3 is done with an extra step, which is a counter for the number of transmission, The Algorithm 3 again checks all n bits of the binary representation of a node in $O(n)$ time and transmit a message along an edge in a constant time. The extra step only takes $O(1)$ time. These are also done for all target nodes so the total time of $O(n^2)$ is needed for doing this step.

- Step 3. Here, for all the nodes with the same distance from the identity node using just hypercube edges or using hypercube and complement edges, the number of 1 bits in their binary representation are saved. The maximum number of such these nodes can be $2n-1$. For these nodes , all n bits of their binary strings are checked to find the 1 bits. So this step can also be done in $O(n^2)$ time.

- Step 4. $O(n^2)$ time is needed to convert the $2n-1$ set of integer values to the binary set representations of length $2n-1$.

- Step 5. As mentioned in [8] finding an SDR can be done in $O(n^{5/2})$.

- Step 6.1. This step takes $O(n^2)$ time to check all even columns of the matrix.

- Step 6.2. For the bits which are 1 in the binary representation of a node, hypercube traversing is the same as traversing in the augmented cube using edges with odd weights computed by $2i-1$. Thus, the column numbers of the bits which are 1 are converted to edges with odd weights calculated by $2i-1$.

This step is same as step 4 so $O(n^2)$ time is needed to convert the maximum $2n-1$ set of integer values to the set of binary representations of length $2n-1$.

- Step 6.3. Here also the time taken to update the matrix is of order $O(n^2)$. Since for each node with both hypercube routing and augmented cube routing, we should go through all $2n-1$ bits in its binary representation of both routings. So $O((2n-1) \times 2 \times (2n-1)) = O(8n^2)$ time is needed.

- Step 6.4. Same as step 5 existence of an SDR is checked which takes $O(n^{5/2})$ time.

- Step 6.6. Same as step 6.3 for each node with both hypercube routing and augmented cube routing, we should go through all $2n-1$ bits in its binary representation and change the bits of the routing which is not used. So this step needs $O((2n-1) \times (2n-1)) = O(4n^2)$ time.

- Step 7. The last step is the same procedure presented in [8] which is an $O(n^3)$ algorithm.

Consequently our algorithm can be done in a total time of $O(n^3)$ which is a polynomial in terms of dimension of the augmented cube.

5.3.3 Example

We end this chapter by providing examples to elaborate on each step of algorithm 4.

Example. In a 6-augmented cube, given eleven target nodes, $t_1, t_2, t_3, ..., t_{11}$, whose binary representation is as follows:

$t_1 : 011010$, $t_2 : 011110$, $t_3 : 101011$, $t_4 : 101110$, $t_5 : 101111$, $t_6 : 110100$, $t_7 : 110110$, $t_8 : 110111$, $t_9 : 111010$, $t_{10} : 111101$, $t_{11} : 100111$

As discussed in previous chapters this number of target nodes is the most possible since $2 \times 6 - 1 = 11$.

Step 1. Considering nodes in a hypercube, Hamming weight of each node is calculated, (Hamming weight of a node is equal to its distance from node 0^n). Result of this step is as follows:

$H_1 : 3$

$H_2 : 4$

$H_3 : 4$

$H_4 : 4$

$H_5 : 5$

$H_6 : 3$

$H_7 : 4$

$H_8 : 5$

$H_9 : 4$

$H_{10} : 5$

$H_{11} : 4$

Step 2. In this step for each target node, a set of weights needed for routing to this node from source node and number of weights in each set are created:

$W_1 : \{4, 7, 11\}, \quad N_1 : 3$

$W_2 : \{4, 11\}, \quad N_2 : 2$

$W_3 : \{1, 5, 10\}, \quad N_3 : 3$

$W_4 : \{1, 6, 11\}, \quad N_4 : 3$

$W_5 : \{1, 6\}, \quad N_5 : 2$

$W_6 : \{2, 5, 10\}, \quad N_6 : 3$

$W_7 : \{2, 5, 11\}, \quad N_7 : 3$

$W_8 : \{2, 5\}, \quad N_8 : 2$

$W_9 : \{2, 7, 11\}, \quad N_9 : 3$

$W_{10} : \{2, 9\}, \quad N_{10} : 2$

$W_{11} : \{1, 8\}, \quad N_{11} : 2$

Step 3. Now H_i is compared with N_i for $1 \leq i \leq 11$. If they are the same, a set is built containing edges with odd weight $2i-1$ where i is the number of bits that are 1.

$H_1 = N_1$, in $t_1 : 011010$, bit numbers $2, 3, 5$ are 1. So weight of odd edges would be: $B_1 : \{3, 5, 9\}$

$H_2 \neq N_2$

$H_3 \neq N_3$

$H_4 \neq N_4$

$H_5 \neq N_5$

$H_6 = N_6$, in $t_6 : 110100$ position of 1s are $1, 2, 4$, so its set of weights would be: $B_6 : \{1, 3, 7\}$

$H_7 \neq N_7$

$H_8 \neq N_8$

$H_9 \neq N_9$

$H_{10} \neq N_{10}$

$H_{11} \neq N_{11}$

Step 4. A binary string of length $2n-1$ is made for each of $2n-1$ target nodes in this step. First consider a zero bit string, 0^{2n-1}, for all $2n-1$ target nodes. Then for each $t_i, 1 \leq i \leq 2n-1$ change bit numbers w_{i_j} of the binary string into 1 starting from left, where $w_{i_j} \in W_i$ $1 \leq j \leq 2n-1$. This step forms a matrix with routing sequences in a binary representation:

$R_1 : 00010010001, R_2 : 00010000001, R_3 : 10001000010, R_4 : 10000100001, R_5 : 10000100000, R_6 : 01001000010, R_7 : 01001000001, R_8 : 01001000000, R_9 : 01000010001, R_{10} : 01000000100, R_{11} : 10000001000$

0	0	0	1	0	0	1	0	0	0	1
0	0	0	1	0	0	0	0	0	0	1
1	0	0	0	1	0	0	0	0	1	0
1	0	0	0	0	1	0	0	0	0	1
1	0	0	0	0	1	0	0	0	0	0
0	1	0	0	1	0	0	0	0	1	0
0	1	0	0	1	0	0	0	0	0	1
0	1	0	0	1	0	0	0	0	0	0
0	1	0	0	0	0	1	0	0	0	1
0	1	0	0	0	0	0	0	1	0	0
1	0	0	0	0	0	0	1	0	0	0

Step 5. There is no SDR for this matrix since all the bits in column 3 are zero.

Step 6.1. All even columns have atleast one non-zero bit and the only column with all zero bits is column 3.

Step 6.3. In this step the hypercube routings, B_1 and B_6, are converted into a bit string with length $2n - 1$.

R'_1 : 00101000100 and R'_6 : 10100010000

Each of these routings are combined with its related line in the matrix.

R_1	0	0	**1**	1	1	0	1	0	1	0	1
R_2	0	0	0	**1**	0	0	0	0	0	0	1
R_3	1	0	0	0	1	0	0	0	0	**1**	0
R_4	1	0	0	0	0	**1**	0	0	0	0	1
R_5	**1**	0	0	0	0	1	0	0	0	0	0
R_6	1	**1**	1	0	1	0	1	0	0	1	0
R_7	0	**1**	0	0	1	0	0	0	0	0	**1**
R_8	0	1	0	0	**1**	0	0	0	0	0	0
R_9	0	1	0	0	0	0	**1**	0	0	0	1
R_{10}	0	1	0	0	0	0	0	0	**1**	0	0
R_{11}	1	0	0	0	0	0	0	**1**	0	0	0

Step 6.4. Bold elements in the above matrix form an SDR.

Step 6.6. In each line that contains both routings, the routing that its bit is used in

the SDR is kept and the other one is deleted. This would result in bellow matrix.

R_1	0	0	**1**	0	1	0	0	0	1	0	0
R_2	0	0	0	**1**	0	0	0	0	0	0	1
R_3	1	0	0	0	1	0	0	0	0	**1**	0
R_4	1	0	0	0	0	**1**	0	0	0	0	1
R_5	**1**	0	0	0	0	1	0	0	0	0	0
R_6	0	**1**	0	0	1	0	0	0	0	1	0
R_7	0	**1**	0	0	1	0	0	0	0	0	**1**
R_8	0	1	0	0	**1**	0	0	0	0	0	0
R_9	0	1	0	0	0	0	**1**	0	0	0	1
R_{10}	0	1	0	0	0	0	0	0	**1**	0	0
R_{11}	1	0	0	0	0	0	0	**1**	0	0	0

Step 7. At last step the algorithm presented in [8] is applied on the matrix to find the disjoint shortest paths.

Bold elements in the above matrix are forming an SDR for $2n-1$ sets. Since $H(R_1) = 3$, the reduction is done on R_1. We first try the 1 in column 5 (from the left) and the

result is a unique set not equal to any other row. so this one is removed to get:

R_1	0	0	**1**	0	0	0	0	0	1	0	0
R_2	0	0	0	**1**	0	0	0	0	0	0	1
R_3	1	0	0	0	1	0	0	0	0	**1**	0
R_4	1	0	0	0	0	**1**	0	0	0	0	1
R_5	**1**	0	0	0	0	1	0	0	0	0	0
R_6	0	**1**	0	0	1	0	0	0	0	1	0
R_7	0	**1**	0	0	1	0	0	0	0	0	**1**
R_8	0	1	0	0	**1**	0	0	0	0	0	0
R_9	0	1	0	0	0	0	**1**	0	0	0	1
R_{10}	0	1	0	0	0	0	0	0	**1**	0	0
R_{11}	1	0	0	0	0	0	0	**1**	0	0	0

We now have a disjoint ordering as follows:

$$O_{R_1} = (5)$$

The next row to reduce is R_3. We remove the first 1 to obtain:

$$
\begin{array}{cccccccccccc}
R_1 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
R_2 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
R_3 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 \\
R_4 & 1 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 1 \\
R_5 & \mathbf{1} & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
R_6 & 0 & \mathbf{1} & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
R_7 & 0 & \mathbf{1} & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} \\
R_8 & 0 & 1 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 \\
R_9 & 0 & 1 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 1 \\
R_{10} & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\
R_{11} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0
\end{array}
$$

And the disjoint ordering now becomes:

$$O_{R_1} = (5)$$
$$O_{R_3} = (1)$$

All reductions are straightforward and we will end up with a matrix which only has the 1s forming the SDR.

R_1	0	0	**1**	0	0	0	0	0	0	0	0
R_2	0	0	0	**1**	0	0	0	0	0	0	0
R_3	0	0	0	0	0	0	0	0	0	**1**	0
R_4	0	0	0	0	0	**1**	0	0	0	0	0
R_5	**1**	0	0	0	0	0	0	0	0	0	0
R_6	0	**1**	0	0	0	0	0	0	0	0	0
R_7	0	**1**	0	0	0	0	0	0	0	0	**1**
R_8	0	0	0	0	**1**	0	0	0	0	0	0
R_9	0	0	0	0	0	0	**1**	0	0	0	0
R_{10}	0	0	0	0	0	0	0	0	**1**	0	0
R_{11}	0	0	0	0	0	0	0	**1**	0	0	0

The final disjoint ordering is:

$$O_{R_1} = (3, 9, 5)$$

$$O_{R_2} = (4, 11)$$

$$O_{R_3} = (10, 5, 1)$$

$$O_{R_4} = (6, 11, 1)$$

$$O_{R_5} = (1, 6)$$

$$O_{R_6} = (2, 10, 5)$$

$$O_{R_7} = (11, 5, 2)$$

$$O_{R_8} = (5, 2)$$

$$O_{R_9} = (7, 11, 2)$$

$$O_{R_{10}} = (9, 2)$$

$$O_{R_{11}} = (8, 1)$$

It can be verified that nodes which are used in routings to all target nodes, are only met once, which means that paths are disjoint. Since each row was a shortest path to each target, so the results are disjoint and shortest paths to target nodes.

Example. In previous example if we change t_{11} to 111110, results are changed as follows:

Step 1. $H_{11} : 5$

Step 2. $W_{11} : \{2, 11\}$ $N_{11} : 2$

Step 3. $H_{11} \neq N_{11}$

Step 4. $R_{11} : 01000000001$

0	0	0	1	0	0	1	0	0	0	1
0	0	0	1	0	0	0	0	0	0	1
1	0	0	0	1	0	0	0	0	1	0
1	0	0	0	0	1	0	0	0	0	1
1	0	0	0	0	1	0	0	0	0	0
0	1	0	0	1	0	0	0	0	1	0
0	1	0	0	1	0	0	0	0	0	1
0	1	0	0	1	0	0	0	0	0	0
0	1	0	0	0	0	1	0	0	0	1
0	1	0	0	0	0	0	0	1	0	0
0	1	0	0	0	0	0	0	0	0	1

As we can see in this matrix, all bits in column 8 are zero. Since it is an even column, none of the hypercube routing of t_1 and t_6 has 1 in their 8-th column. This means that we cannot have an SDR for these set of targets so there would be no disjoint and shortest paths from source node to these eleven target nodes.

Chapter 6

Conclusion

In this thesis, we have studied the (n, k)-arrangement graph, an attractive alternative to the n-star network. We found some useful topological properties of this graph. $A_{n,k}$ is a $k(n-k)$-regular graph. We proved there exists $k(n-k)$ disjoint paths for sending a message from a single source to $k(n-k)$ targets with lengths no more than $diameter + (n-k) + 2$. For each of $k(n-k)$ neighbors of the identity node, its foreign element is fixed in its position during routing, which will result in finding disjoint paths. Also we used a similar procedure presented in [26] for finding n disjoint paths between two nodes, where $1 < k < n-1$. This routing paradigm takes $O(n^2)$ time to find all n paths. The shortest distance between two nodes is shown by $d(s,t)$ and the length of n disjoint paths found in this thesis is at most $d(s,t)+4$. Moreover, a new variation of the hypercube called the augmented cubes, AQ_n, are studied as well. The number of nodes and their structure in an augmented cube are the same as the hypercube, but an AQ_n is a $(2n-1)$-regular graph. For taking advantage of this property, we have designed an algorithm to find shortest and disjoint paths for sending a message from a single source node to the greatest possible number of target nodes, which is $2n-1$. The proposed algorithm has time complexity of $O(n^3)$ that is a polynomial in terms of the dimension of the graph.

One of the future directions would be to see whether the $k(n-k)$ paths in the arrangement graph can have a constant bound. Clearly for finding $k(n-k)$ disjoint paths in an (n,k)-arrangement graph an algorithm can be designed. The first step is to consider its time complexity and try to design an algorithm with reasonable time taken to find results.

As for the augmented cube graph, the algorithm can be modified by reducing the time taken to find all disjoint paths. Studies should be done to figure out if it is possible to find these paths without checking the matrix for replacing a routing with an alternative hypercube path, if there exists any.

Bibliography

[1] Sheldon B. Akers and Balakrishnan Krishnamurthy. A group-theoretic model for symmetric interconnection networks. *IEEE transactions on Computers*, 38(4):555–566, 1989.

[2] Selim G Akl. *Parallel computation: models and methods*. Prentice-Hall, Inc., 1997.

[3] Abdel-Elah Al-Ayyoub and Khaled Day. The hyperstar interconnection network. *Journal of Parallel and Distributed Computing*, 48(2):175–199, 1998.

[4] Jeffe Boats, Lazaros Kikas, and John Olesik. An algebraic approach for finding disjoint paths in the alternating group graph. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 64:109, 2008.

[5] Jou-Ming Chang, Jinn-Shyong Yang, Yue-Li Wang, and Yuwen Cheng. Panconnectivity, fault-tolerant Hamiltonicity and Hamiltonian-connectivity in alternating group graphs. *Networks*, 44(4):302–310, 2004.

[6] Eddie Cheng, Jerrold W Grossman, Ke Qiu, and Zhizhang Shen. The number of shortest paths in the arrangement graph. *Information Sciences*, 240:191–204, 2013.

[7] Eddie Cheng, Ke Qiu, and Zhi Zhang Shen. On disjoint shortest paths routing in interconnection networks: A case study in the star graph, 2012.

[8] Eddie Cheng, Ke Qiu, and Zhizhang Shen. A faster algorithm for finding disjoint ordering of sets. *International Journal of Networking and Computing*, 3(2):182–191, 2013.

[9] Wei-Kuo Chiang and Rong-Jaye Chen. The (n, k)-star graph: A generalized star graph. *Information Processing Letters*, 56(5):259–264, 1995.

[10] Sheshayya A Choudum and V Sunitha. Augmented cubes. *Networks*, 40(2):71–84, 2002.

[11] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. Introduction to algorithms second edition, 2001.

[12] Khaled Day and Anand Tripathi. Arrangement graphs: a class of generalized star graphs. *Information Processing Letters*, 42(5):235–241, 1992.

[13] Michael J Flynn. Some computer organizations and their effectiveness. *IEEE transactions on computers*, 100(9):948–960, 1972.

[14] Shuhong Gao, Beth Novick, and Ke Qiu. From hall's matching theorem to optimal routing on hypercubes. *Journal of Combinatorial Theory, Series B*, 74(2):291–301, 1998.

[15] Philip Hall. On representatives of subsets. *Journal of the London Mathematical Society*, 1(1):26–30, 1935.

[16] John E Hopkroft and Richard M Karp. An $n^{5/2}$ algorithm for maximum matching in bipartite graphs. *SIAM J. Comput*, 2:225–231, 1973.

[17] Jung-Sing Jwo, S Lakshmivarahan, and Sudarshan K Dhall. A new class of interconnection networks based on the alternating group. *Networks*, 23(4):315–326, 1993.

[18] Cheng-Nan Lai. An optimal construction of node-disjoint shortest paths in hypercubes. In *Proceedings of the 28 th Workshop on Combinatorial Mathematics and Computation Theory*, pages 245–253, 2011.

[19] Sivaramakrishnan Lakshmivarahan, Jung-Sing Jwo, and Sudarshan K. Dhall. Symmetry in interconnection networks based on cayley graphs of permutation groups: A survey. *Parallel Computing*, 19(4):361–407, 1993.

[20] Hyeong-Ok Lee, Jong-Seok Kim, Eunseuk Oh, and Hyeong-Seok Lim. Hyper-star graph: A new interconnection network improving the network cost of the hypercube. *EurAsia-ICT 2002: Information and Communication Technology*, pages 858–865, 2002.

[21] Jingli Li, Yonghong Xiang, Manli Chen, and Yongheng Zhou. Broadcasting in (n, k)-arrangement graph based on an optimal spanning tree. In *Modelling & Simulation, 2007. AMS'07. First Asia International Conference on*, pages 193–197. IEEE, 2007.

[22] Yifeng Li. Properties algorithms of the $(n, k)-$arrangement graphs. *Master Thesis, Department of Computer Science Brock University*, 2009.

[23] Chin-Tsai Lin and Wen-Chuan Chiu. A routing scheme for constructing node-to-node disjoint paths in alternating group graphs. In *Proc. 19th Workshop on Combinatorial Math. and Computat. Theory*, pages 89–99, 2002.

[24] Meijie Ma, Guizhen Liu, and Jun-Ming Xu. Panconnectivity and edge-fault-tolerant pancyclicity of augmented cubes. *Parallel Computing*, 33(1):36–42, 2007.

[25] Franco P Preparata and Jean Vuillemin. The cube-connected cycles: a versatile network for parallel computation. *Communications of the ACM*, 24(5):300–309, 1981.

[26] Ke Qiu and Selim G Akl. On node-to-node disjoint paths in the star interconnection network. In *IASTED PDCS*, pages 731–735, 2005.

[27] Youcef Saad and Martin H Schultz. Topological properties of hypercubes. *IEEE Transactions on computers*, 37(7):867–872, 1988.

[28] Jianping Song, Zifeng Hou, and Yuntao Shi. An optimal multicast algorithm for cube-connected cycles. *Journal of Computer Science and Technology*, 15(6):572–583, 2000.

[29] Nian-Feng Tzeng and Sizheng Wei. Enhanced hypercubes. *IEEE Transactions on Computers*, 40(3):284–294, 1991.

[30] Fan Zhang, Ke Qiu, and Jong Seok Kim. Hyper-star graphs: Some topological properties and an optimal neighbourhood broadcasting algorithm. *Concurrency and Computation: Practice and Experience*, 27(16):4186–4193, 2015.